RUBRICS ON TRIAL: EVOLVING RUBRICS FROM A SINGLE QUERY VIA SYNTHETIC PAIRWISE EVIDENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Rubric evolution offers a promising approach to improving the quality of rubrics generated by large language models (LLMs). Central to this process is rubric comparison, which identifies the better of two rubrics and guides the direction of evolution. However, accurate rubric comparison is difficult, which presents two challenges. ❶ **It should reflect downstream task performance**, which is essential for assessing rubric utility but often prohibitively expensive to evaluate. ❷ **It should discourage unnecessary criteria**, which increase verification costs and may dilute the influence of essential criteria. To address these challenges, we introduce **Rubrics on Trial**, a multi-agent framework that evolves rubrics by comparing synthetic response pairs. To address challenge ❶, the framework compares synthetic responses that satisfy the respective rubrics, providing a proxy for downstream performance without training a separate policy for each rubric. To address challenge ❷, it assesses the necessity of a candidate criterion by independently generating high-quality alternative responses that violate it and comparing them with edited versions that satisfy it. A rubric is favored when it improves response quality in both comparisons, and the resulting comparison signal is further incorporated for rubric evolution. Extensive experiments demonstrate that Rubrics on Trial improves the quality of generated rubrics and leads to better downstream task performance. Code is available at https://anonymous.4open.science/r/Trial-9464.

1 INTRODUCTION

A rubric is a structured set of criteria specifying the requirements an LLM response should meet to satisfy a user query (Hashemi et al., 2024). By translating user intent into explicit criteria, rubrics enable fine-grained, verifiable assessment of LLM responses. These properties make rubrics well suited for providing reward signals in reinforcement learning from human feedback (RLHF) (Gunjal et al., 2026). Currently, rubrics have been a critical component for LLM evaluation and training.

Early efforts to construct rubrics relied on human annotators to formulate evaluation criteria (Arora et al., 2025). Although effective, it is difficult to extend across diverse user queries and tasks. Subsequent studies used LLMs to generate query-specific rubrics directly, providing a scalable alternative (Cook et al., 2024; Wei et al., 2025). Nevertheless, these approaches may overlook important criteria and do not inherently support iterative refinement of the resulting rubrics (Dhole & Agichtein, 2026). To address this limitation, recent work has explored rubric evolution (Zhang et al., 2026; Shen et al., 2026), drawing on the self-improvement capabilities of LLMs to refine generated rubrics through successive rounds of revision and comparison. This approach offers a promising way to obtain high-quality rubrics without requiring human annotation (Mahmoud et al., 2026).

Accurate rubric comparison is central to rubric evolution, since it determines which revisions are retained and thereby guides the direction of evolution. However, this task is difficult even for human experts, which presents two critical challenges. ❶ **Rubric comparison should reflect downstream task performance.** The practical value of a rubric depends on its ability to improve response quality after RLHF. Yet directly assessing this value by training and evaluating a policy with each candidate rubric is often computationally prohibitive. ❷ **Rubric comparison should discourage unnecessary criteria.** A criterion may appear reasonable but does not improve response quality. Such criteria

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.

increase verification costs and may dilute the influence of essential requirements (Mahmoud et al., 2026), which should be identified and discouraged in rubric comparison process.

To address these challenges, we introduce **Rubrics on Trial**, a multi-agent framework that evolves rubrics through comparisons of synthetic response pairs. To address challenge ❶, it generates and compares responses that satisfy the respective candidate rubrics. This comparison serves as a proxy for assessing each rubric's potential to improve response quality after RLHF, without requiring training a separate policy for every candidate. To address challenge ❷, it generates high-quality alternative responses that violate a candidate criterion, then compares these with edited versions that satisfy the criterion, to assess whether the criterion is necessary for improving response quality. A rubric is favored only when both comparisons favor it, and the comparison result guides rubric evolution.

Our contributions can be summarized as follows. ❶ **We identify two critical challenges in rubric comparison,** which should reflect downstream task performance while discouraging unnecessary criteria. ❷ **We introduce Rubrics on Trial, a multi-agent framework for rubric comparison and evolution.** It addresses the identified challenges through synthetic response pairs and alternative responses, and provides a theoretical guarantee for the surrogate comparison strategy. **We demonstrate the effectiveness of the proposed method through extensive experiments.** Our method outperforms existing rubric generation approaches, and evolution effectively improves the quality of generated rubrics. These improvements translate into better downstream RLHF performance.

2 Preliminaries

2.1 Rubric-based Evaluation

Rubric-based evaluation assesses response quality against explicit criteria (Hashemi et al., 2024). Given a user query $x \in \mathcal{X}$ and an LLM-generated response $y \in \mathcal{Y}$, we define a rubric $r = \{c_1, c_2, \ldots, c_{\mathrm{K}}\}$ comprising K readily verifiable criteria. Each criterion is a textual statement characterizing a property of an ideal response. Suppose $v(\cdot)$ is an LLM-based verifier that assesses whether the response y satisfies the criterion c_k given x, the evaluation score is defined as:

$$s(x, y) = \frac{1}{\mathrm{K}} \sum_{k=1}^{\mathrm{K}} v(x, y, c_k), \tag{1}$$

which aggregates verification results across all criteria. $s(x, y)$ provides a comprehensive evaluation of LLM response and supports various downstream applications, such as selecting high-quality responses (Li et al., 2026) and providing reward signal in RLHF (Gunjal et al., 2026).

2.2 Rubric Generation and Evolution

Rubric generation is a long-standing and formidable challenge. Some approaches relied on expert annotation (Arora et al., 2025), which is costly and difficult to scale. To address this limitation, other methods employ LLMs to generate rubrics directly from user queries (Cook et al., 2024; Wei et al., 2025). The premise is that LLMs can induce what an ideal response should encompass to comprehensively address the query. This approach enables efficient rubric construction at scale, with numerous applications in real-world practices (Dhole & Agichtein, 2026).

The self-improvement capabilities of LLMs render it feasible to evolutionarily optimize the quality of LLM-generated rubrics. Specifically, let r_t denote the rubric at round t, a single evolution step for r_t can be formulated as follows:

$$\hat{r}_{t+1} = \mathrm{Propose}(x, r_t), \quad \hat{d}_t = \mathrm{Compare}(x, r_t, \hat{r}_{t+1}), \quad r_{t+1} = \mathrm{Update}(x, r_t, \hat{r}_{t+1}, \hat{d}_t), \tag{2}$$

which starts from generating a candidate rubric $\hat{r}_{t+1}$ as a variant of r_t. A binary comparison signal $\hat{d}_t$ is then yielded, where $\hat{d}_t = 1$ indicates that $\hat{r}_{t+1}$ is superior to r_t, and $\hat{d}_t = 0$ otherwise. The update follows a simple greedy strategy: r_{t+1} retains the original r_t if $\hat{d}_t = 0$, and adopts the candidate $\hat{r}_{t+1}$ if $\hat{d}_t = 1$. The quality of r_t is expected to improve as t increases. Rubric evolution has recently attracted increasing research attention. Zhang et al. (2026) elicits discriminative rubrics by contrasting high-quality, diverse responses in the high-reward region; Shen et al. (2026) recursively decompose coarse criteria into finer-grained ones and filter out redundant criteria; Xu et al. (2026b) alternate rubric-memory refinement with adversarial updates to uncover supplementary criteria.

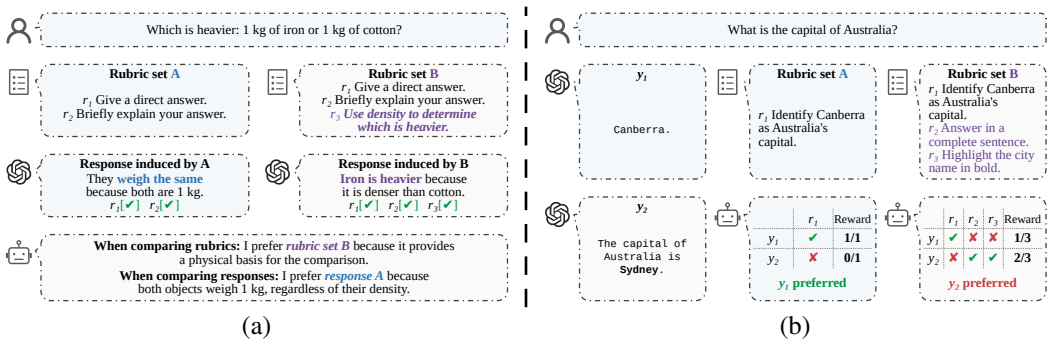


Figure 1: Two case studies illustrating challenges in rubric comparison. (a) Direct comparison of rubric texts favors rubric B, although rubric A induces a better response. (b) Unnecessary criteria (r_2 and r_3) cause the incorrect response (y_2) to receive a higher score than the correct response (y_1).

3 METHODOLOGY

3.1 MOTIVATION

Accurate rubric comparison is central to evolving LLM-generated rubrics toward higher quality. It determines whether a candidate rubric outperforms the current rubric (i.e., $\hat{r}_{t+1} \succ r_t$), which guides the direction of evolution. Inaccurate rubric comparison can misdirect the evolutionary process and ultimately undermine the quality of LLM-generated rubrics. Therefore, accurate comparison is critical to consistently improve rubric quality during evolution.

However, accurate rubric comparison is non-trivial: even human experts may struggle to judge whether one rubric outperforms another. This difficulty stems from two key challenges. ❶ **Rubric comparison should reflect downstream task performance.** Specifically, rubrics are often used to provide reward signals in RLHF; consequently, a high-quality rubric should lead to improved model responses after RLHF. However, evaluating such downstream performance requires executing the full RLHF pipeline, which is often prohibitively expensive. Therefore, comparing rubrics based on downstream performance is important but difficult. ❷ **Rubric comparison should discourage unnecessary criteria.** In practice, unnecessary criteria often exist in LLM-generated rubrics (Mahmoud et al., 2026), which refer to criteria not essential for satisfying the user query. They not only increase verification costs, but may also dilute the influence of truly necessary criteria. Therefore, rubric comparison should favor rubrics that preserve only necessary ones.

Case studies. Figure 1 illustrates these challenges through two case studies. For challenge ❶, panel (a) presents a scenario in which direct comparison of rubric texts fails to reflect downstream performance. Although text comparison favors rubric B, rubric A induces a better response: it correctly states that both objects weigh 1 kg, For challenge ❷, panel (b) shows how unnecessary criteria can distort response rankings. Under rubric B, response y_2 violates the correctness criterion (r_1) but satisfies the style-related criteria (r_2 and r_3). Consequently, it receives a higher score than y_1, which correctly answers "Canberra" satisfying the user query but violates the style-related criteria.

Given the importance of accurate rubric comparison for rubric evolution and the challenges outlined above, we seek a rubric comparison strategy that addresses both challenges. This motivates three research questions: *(1) How can rubric comparison reflect downstream RLHF performance? (2) How can rubric comparison discourage criteria that are unnecessary for satisfying the user query? (3) Does it improve rubric evolution and the effectiveness of the resulting rubrics?*

3.2 IMPROVING RUBRIC COMPARISON VIA SYNTHETIC RESPONSE PAIRS

To account for downstream RLHF performance in rubric comparison (Challenge ❶), a direct approach is to evaluate each rubric through the policy it produces under RLHF. Given the current rubric r_t and the candidate rubric $\hat{r}_{t+1}$, we independently apply RLHF to the same initial policy π, using the scores assigned by each rubric as rewards, to obtain policies π_{r_t} and $\pi_{\hat{r}_{t+1}}$, respectively.

For a given query x, we evaluate each policy in terms of its expected response quality:

$$\mu(r_t) = \mathbb{E}_{y\sim\pi_{r_t}(\cdot|x)}[g(x,y)], \quad \mu(\hat{r}_{t+1}) = \mathbb{E}_{y\sim\pi_{\hat{r}_{t+1}}(\cdot|x)}[g(x,y)], \tag{3}$$

where $g(x, y)$ is an independent measure of response quality. In practice, $\mu(r_t)$ can be estimated by sampling responses from π_{r_t}, scoring them with an LLM judge, and averaging the resulting scores. We prefer r_t to $\hat{r}_{t+1}$, denoted by $r_t \succ \hat{r}_{t+1}$, if $\mu(r_t) > \mu(\hat{r}_{t+1})$, indicating that r_t yields a policy with higher expected response quality after RLHF. However, this approach requires training a separate policy for every candidate rubric, making it prohibitively expensive for rubric comparison.

To reduce computational cost, we propose a surrogate comparison strategy that uses responses sampled from the initial policy π. Motivated by the expectation that RLHF encourages responses to satisfy the reward rubric, we approximate the post-RLHF response distribution by conditioning π on rubric satisfaction. The resulting surrogates for $\mu(r_t)$ and $\mu(\hat{r}_{t+1})$ are:

$$\hat{\mu}(r_t) = \mathbb{E}_{y\sim\pi(\cdot|x)}\big[g(x,y) \mid y \models r_t\big], \qquad \hat{\mu}(\hat{r}_{t+1}) = \mathbb{E}_{y\sim\pi(\cdot|x)}\big[g(x,y) \mid y \models \hat{r}_{t+1}\big], \tag{4}$$

where $y \models r$ indicates that y satisfies all criteria in r. These quantities can be estimated by sampling responses from $\pi(\cdot \mid x)$ and, for each rubric, averaging the quality scores of the responses that satisfy it. We then prefer $r_t \succ \hat{r}_{t+1}$, if $\hat{\mu}(r_t) > \hat{\mu}(\hat{r}_{t+1})$. This strategy eliminates the need for RLHF during rubric comparison. Besides, it preserves the comparison preference obtained by evaluating the post-RLHF policies under mild conditions (see Theorem 1). Therefore, this strategy provides an efficient approach to rubric comparison that accounts for downstream RLHF performance.

Theorem 1. *Assume that rubric satisfaction has positive probability under the initial policy π, so that $\hat{\mu}$ is well-defined, and each post-RLHF policy fully satisfies its reward rubric:*

$$\mathbb{P}_{y\sim\pi_{r_t}(\cdot|x)}(y \models r_t) = 1, \qquad \mathbb{P}_{y\sim\pi_{\hat{r}_{t+1}}(\cdot|x)}(y \models \hat{r}_{t+1}) = 1. \tag{5}$$

Let $\beta(r) = \mu(r) - \hat{\mu}(r) \geq 0$ denote the quality gain from RLHF for $r \in \{r_t, \hat{r}_{t+1}\}$. If $\hat{\mu}(r_t) - \hat{\mu}(\hat{r}_{t+1}) > \beta(\hat{r}_{t+1}) - \beta(r_t)$, then $\hat{\mu}(r_t) > \hat{\mu}(\hat{r}_{t+1})$ implies $\mu(r_t) > \mu(\hat{r}_{t+1})$.

The implementation of the surrogate comparison strategy is detailed as follows. At round t, we construct the candidate by adding a criterion c:

$$\hat{r}_{t+1} = r_t \cup \{c\}. \tag{6}$$

Given a query x, we generate a response a^+ that satisfies all criteria in $\hat{r}_{t+1}$. We then minimally edit a^+ to obtain a^-, which violates c but satisfies all criteria in r_t. The minimal editing here aims to isolate the impact of c by removing unrelated changes. To enforce the rubric-satisfaction constraints, we sample multiple responses from π and retain those that pass the following checks by a verifier v:

$$\begin{aligned} v(x, a^+, c) &= 1, \qquad v(x, a^-, c) = 0, \\ v(x, a^+, c_k) &= 1, \qquad v(x, a^-, c_k) = 1, \quad \forall c_k \in r_t. \end{aligned} \tag{7}$$

We then use an LLM comparator to compare a^+ and a^- given x. If a^+ is favored, we derive $\hat{r}_{t+1} \succ r_t$. This strategy provides an efficient approach to rubric comparison while accounting for downstream RLHF performance.

Table 1: Rubric comparison accuracy.

Compare	RubricBench		HealthBench		ResearchQA	
	ACC	Δ	ACC	Δ	ACC	Δ
Comparator: DeepSeek-V4-Flash						
Rubric text	0.72	-	0.40	-	0.38	-
Responses	**0.84**	**16.67%↑**	**0.91**	**126.25%↑**	**0.73**	**90.79%↑**
Comparator: Gemini-3.1-Flash-Lite						
Rubric text	0.36	-	0.15	-	0.34	-
Responses	**0.77**	**113.89%↑**	**0.76**	**420.69%↑**	**0.65**	**91.18%↑**

Note: Δ indicates the relative improvement over rubric-text comparison in percentage. The best results are **bolded**.

Case study. To investigate whether synthetic responses improve the accuracy of rubric comparison. We use an LLM to add a plausible but harmful criterion to standard rubric in public benchmarks. Two LLMs then identify the better rubric by comparing either the rubric texts or their corresponding synthetic responses (see Appendix D.2 for details). As shown in Table 1, response comparison identifies the better rubric more accurately across both LLMs and all three benchmarks, supporting the use of synthetic responses for rubric comparison.

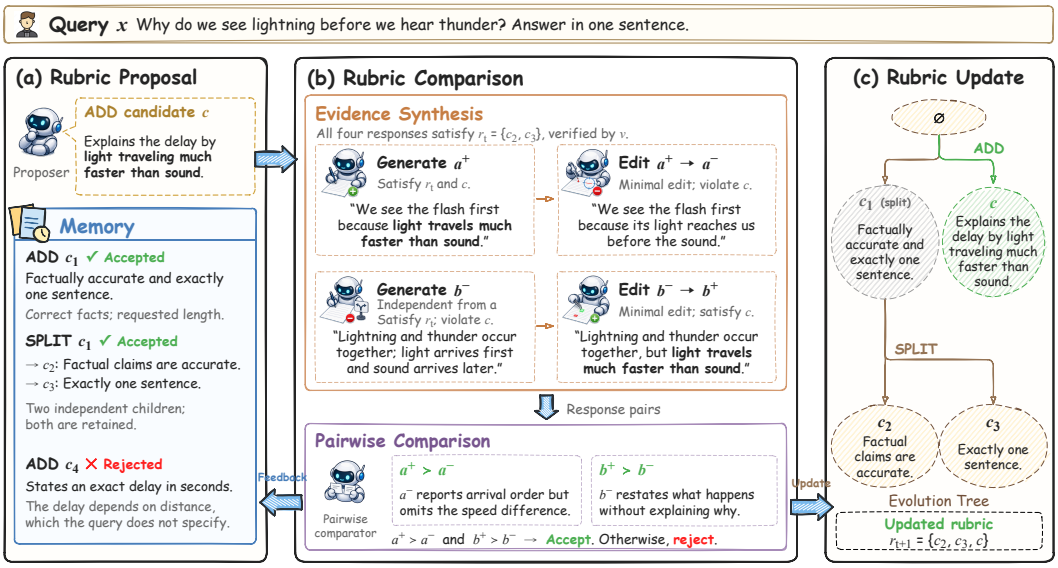


Figure 2: The overview of Rubrics on Trial for one round of evolution.

3.3 Discouraging Unnecessary Criteria with Alternative Responses

Although comparing a^+ and a^- accounts for downstream RLHF performance, it cannot reliably discourage unnecessary criteria (Challenge ❷). The reason is that a^- is constructed by editing a^+ specifically to violate c. In practice, such edits often remove the content associated with c entirely, producing a degraded response rather than a response that satisfies the user query in another way. Consequently, the comparator may favor a^+ even when c is unnecessary to satisfy the user query.

More specifically, consider the query "Which is a prime number? (A) 21 (B) 27 (C) 29" with the current rubric $r_t = \{c_1\}$, where c_1 is "Select (C) 29." Suppose the additional criterion c is "Check the options in at least two separate reasoning steps." The response a^+ satisfies c by first showing that 21 and 27 are composite through factorization and then establishing that 29 is prime. Editing a^+ to violate c removes both reasoning steps, leaving only "Answer: (C)" in a^-. The comparator favors a^+ because a^- gives no justification. However, c is unnecessary: a single-step response such as "29 is the only option not divisible by 3, so the answer is (C)" also provides a valid justification for selecting the correct option. Thus, the comparator may favor a^+ when c is an unnecessary criterion.

To address this limitation, we introduce comparison with alternative responses. The key idea is that an unnecessary criterion can be violated with little or no loss in response quality, since the user query can be satisfied through an alternative approach. We therefore test whether satisfying c improves the quality of a strong response independently generated to violate c. Specifically, given a query x, we generate an alternative response b^- of the highest possible quality, subject to violating c while satisfying all criteria in r_t. Unlike a^-, b^- is generated from scratch rather than edited from a response constructed to satisfy c. This allows b^- to address the user query bypassing c. We then minimally edit b^- to obtain b^+, which satisfies both c and r_t. To enforce rubric-satisfaction constraints, we retain responses from π that pass the following checks:

$$\begin{aligned} v(x, b^+, c) &= 1, \qquad v(x, b^-, c) = 0, \\ v(x, b^+, c_k) &= 1, \qquad v(x, b^-, c_k) = 1, \quad \forall c_k \in r_t. \end{aligned} \tag{8}$$

We then use the LLM comparator to assess the relative quality of b^+ and b^- given x. A preference for b^+ suggests that satisfying c improves even a strong alternative response, providing evidence for the necessity of c. Conversely, a tie or a preference for b^- suggests that the user query can be satisfied at least as well without c, providing evidence that c is unnecessary. In the above example, the single-step response serves as b^-, and editing b^- to check the options in two separate steps yields b^+. Since both responses select (C) with a correct justification, the comparator finds b^+ and b^- equally good, so c is identified as unnecessary. The complete case is presented in Appendix C.2.

Combining the two comparison strategies, we define the rubric comparison signal as

$$\hat{d}_t = \mathrm{Compare}(x, r_t, \hat{r}_{t+1}) = \begin{cases} 1, & a^+ \succ a^- \text{ and } b^+ \succ b^-, \\ 0, & \text{otherwise}, \end{cases} \tag{9}$$

where $\hat{d}_t = 1$ only when $\hat{r}_{t+1}$ improves response quality in both comparisons. This comparison signal reflects downstream RLHF performance while discouraging unnecessary criteria.

3.4 The Workflow of Rubrics on Trial

In this section, we introduce Rubrics on Trial, a multi-agent framework that uses the comparison signal in (9) to guide rubric evolution. As illustrated in Figure 2, the framework comprises a rubric proposer, response generators, a verifier, and a pairwise comparator. These agents iteratively refine rubrics through the proposal, comparison, and update procedure introduced in Section 2.2. Algorithm 1 summarizes the workflow, with the core steps described below.

Rubric proposal. Starting from an empty rubric $r_0 = \varnothing$, the proposer uses the query, the current rubric, and memory $\mathcal{H}$ to propose an ADD or SPLIT operation. ADD introduces an additional criterion c, while SPLIT decomposes a criterion c_p into finer criteria c_i.

Rubric comparison and update. For ADD, we compare r_t and $\hat{r}_{t+1}$ using the comparison signal $\hat{d}_t$ defined in (9). If $\hat{d}_t = 1$, we set $r_{t+1} = \hat{r}_{t+1}$; otherwise, we retain r_t. For SPLIT, we first remove the parent criterion c_p to obtain a shared baseline $r_t \setminus c_p$. Each child criterion c_i defines a candidate rubric

$$\hat{r}_{t+1}^{(i)} = (r_t \setminus \{c_p\}) \cup \{c_i\}.$$

We then compare each candidate independently against the shared baseline to obtain the corresponding signal $\hat{d}_t^{(i)}$. A child criterion c_i is accepted if $\hat{d}_t^{(i)} = 1$. The updated rubric r_{t+1} consists of the baseline criteria and all accepted child criteria.

Algorithm 1 The workflow of Rubrics on Trial.

Input: x: user query; T: maximum evolution rounds; P: patience.
Output: r^*: evolved rubric.

$r_0 \leftarrow \varnothing$; $\mathcal{H} \leftarrow \varnothing$; $u \leftarrow 0$
for $t = 0, \ldots, T-1$ **do**
  $\hat{r}_{t+1} \leftarrow \mathrm{Propose}(x, r_t, \mathcal{H})$
  if ADD **then**
    $\hat{d}_t \leftarrow \mathrm{Compare}(x, r_t, \hat{r}_{t+1})$
    $r_{t+1} \leftarrow \mathrm{Update}(x, r_t, \hat{r}_{t+1}, \hat{d}_t)$
  else if SPLIT **then** $r_{t+1} \leftarrow \varnothing$
    for all children c_i of c_p **do**
      $\hat{d}_t^{(i)} \leftarrow \mathrm{Compare}\big(x, r_t \setminus \{c_p\}, \hat{r}_{t+1}^{(i)}\big)$
      $r_{t+1} \leftarrow r_{t+1} \cup \mathrm{Update}\big(x, r_t \setminus \{c_p\}, \hat{r}_{t+1}^{(i)}, \hat{d}_t^{(i)}\big)$
  $\mathcal{H} \leftarrow \mathcal{H} \cup \{(\hat{r}_{t+1}, r_{t+1}, \text{feedback})\}$
  $u \leftarrow u + 1$ **if** $r_{t+1} = r_t \neq \varnothing$, **else** 0
  if $u \geq P$ **then**
    break
$r^* \leftarrow r_{t+1}$

The memory $\mathcal{H}$ stores historical information to guide subsequent proposals. These steps are iterated for T rounds, stopping early when a nonempty rubric remains unchanged for P consecutive rounds.

4 Experiments

4.1 Experimental Setup

Datasets and Evaluation. We conduct empirical evaluations on JudgeBench (Tan et al., 2025), RM-Bench (Liu et al., 2025), and RewardBench Chat-Hard (Lambert et al., 2025), covering nine evaluation sets. We use each generated rubric to rank benchmark responses by their evaluation scores $s(x, y)$. We report preference accuracy, with half credit for ties, and its unweighted mean across the nine sets (Macro_9). Higher accuracy indicates better discrimination of response quality.

Baselines. We compare against two groups of methods: (1) **LLM-based rubric generation methods**, which prompt fixed LLMs to generate rubrics without training, including the single-pass methods direct generation (Direct), TICK (Cook et al., 2024), and RocketEval (Wei et al., 2025), and the rubric evolution methods RRD-Uniform (Shen et al., 2026) and ChaseTail (Zhang et al., 2026); and (2) **Trained rubric generators**, which are dedicated models trained to generate rubrics, including RubricRM (Liu et al., 2026), RubricARM (Xu et al., 2026a), and RubricARROW (Jiang et al., 2026). All methods generate rubrics from the query without access to the benchmark responses.

Implementation Details. Unless otherwise specified, we use DeepSeek-V4-Flash for rubric generation, response generation and editing, and pairwise comparison. All methods use GPT-4.1-mini as

Table 2: Pairwise preference accuracy (%) across nine evaluation sets from three benchmark suites.

	JudgeBench				RM-Bench				RewardBench	Macro
Method	Knowledge	Reasoning	Math	Coding	Chat	Math	Code	Safety	Chat-Hard	Avg.
LLM-based rubric generation methods										
Direct	67.53	62.08	75.56	67.12	63.01	79.63	65.40	86.09	73.25	71.07
TICK	67.86	62.75	72.78	68.49	62.96	82.52	66.93	87.69	72.81	71.64
RocketEval	69.97	70.47	77.22	68.49	63.48	83.29	66.98	87.55	71.93	73.26
RRD-Uniform	72.56	72.82	82.22	66.44	58.83	81.06	66.86	76.37	66.78	71.55
ChaseTail	71.59	84.23	82.22	**69.86**	63.05	85.55	68.57	87.02	72.04	76.01
Trained rubric generators										
RubricRM	58.60	58.05	70.00	66.44	65.37	67.07	69.20	76.73	71.60	67.01
RubricARM	57.95	58.72	67.78	68.49	66.32	67.31	69.13	75.90	**77.95**	67.73
RubricARROW	53.90	57.38	66.11	67.81	66.75	63.97	65.91	67.90	75.99	65.08
Rubrics on Trial	**74.03**	**89.60**	**85.56**	68.49	**67.96**	**91.27**	**69.66**	**89.64**	76.97	**79.24**

Note: Macro Avg. denotes the mean across all evaluation sets. The best metrics in each column are **bolded**.

the final verifier and uniform criterion weights to isolate rubric quality. Rubrics on Trial runs for at most $T = 10$ rounds with early stopping patience $P = 5$. Further details appear in Appendix D.

4.2 Overall Performance

In this section, we compare Rubrics on Trial with competitive LLM-based rubric generation methods and trained rubric generators. The results are presented in Table 2 with key observations as follows. ❶ **Single-pass rubric generation offers limited gains.** TICK and RocketEval improve only slightly over Direct. This indicates that generating rubrics in a single pass, even with carefully designed prompts, brings limited improvement in rubric quality. ❷ **Rubric evolution is effective to improve rubric quality.** For example, ChaseTail achieves the strongest baseline performance. However, RRD-Uniform barely improves over Direct. This suggests that rubric evolution is promising, but its gains remain unstable without accurate rubric comparison. ❸ **Trained rubric generators transfer unevenly across evaluation sets.** They perform well on chat-oriented evaluation sets such as RM-Bench Chat, but underperform Direct on most other sets, leading to a lower Macro_9 than Direct. This suggests that, despite the training cost, a dedicated rubric generator does not ensure rubric quality across evaluation sets. ❹ **Rubrics on Trial achieves the best rubric quality.** It attains the highest Macro_9 and ranks first on seven of nine evaluation sets, outperforming Direct and the strongest baseline, ChaseTail, by 8.17 and 3.23 percentage points, respectively. This success is attributed to its comparison strategy, which facilitates evolution and improves the quality of generated rubrics.

4.3 Ablation Study

In this section, we dissect the individual contributions of the components within Rubrics on Trial. In Table 3, Rubrics on Trial† replaces iterative rubric evolution with a single revision round, in which the proposer may modify any criteria in the rubric generated by Direct. Rubrics on Trial‡ retains rubric evolution but compares rubric texts rather than synthetic response pairs. The key observations are as follows. ❶ **Rubric evolution benefits the quality of rubrics.** Rubrics on Trial† underperforms the full method on all nine evaluation sets (e.g., Macro_9 drops from 79.24% to 73.41%), validating the importance of multiple rounds of proposal and comparison. ❷ **Synthetic response pairs improve the performance of rubric comparison.** Rubrics on Trial‡ underperforms the full method on eight of nine evaluation sets (e.g., Macro_9 drops to 74.28%), validating the importance of comparing rubrics through the responses that satisfy them.

In Table 4, Rubrics on Trial† and Rubrics on Trial‡ remove (a^+, a^-) and (b^+, b^-), respectively, and compare rubrics with the remaining pair alone. The key observations are as follows. ❶ **Alternative responses help discourage unnecessary criteria.** Rubrics on Trial‡ underperforms the full method on eight of nine evaluation sets (e.g., Macro_9 drops from 79.24% to 76.11%), validating the importance of testing whether an additional criterion improves a strong response that violates it. ❷ **The two response pairs provide complementary benefits.** The full method outperforms both variants on eight of nine evaluation sets, showing that neither pair alone suffices.

Table 3: Ablation study on rubric evolution and synthetic response pairs.

Method	Evol.	Resp.	JudgeBench				RM-Bench				Reward	Macro
			Know.	Reason.	Math	Code	Chat	Math	Code	Safety	Chat-Hard	Avg.
Direct	✗	✗	67.53	62.08	75.56	67.12	63.01	79.63	65.40	86.09	73.25	71.07
Rubrics on Trial†	✗	✓	69.32	70.47	78.89	67.81	62.62	84.32	66.76	86.53	74.01	73.41
Rubrics on Trial‡	✓	✗	70.78	76.17	80.56	**69.18**	63.09	82.12	67.45	86.17	73.03	74.28
Rubrics on Trial	✓	✓	**74.03**	**89.60**	**85.56**	68.49	**67.96**	**91.27**	**69.66**	**89.64**	**76.97**	**79.24**

Note: Macro Avg. denotes the mean across all evaluation sets. The best and second-best metrics are **bolded** and underlined, respectively.

Table 4: Ablation study on the two response pairs (a^+, a^-) and (b^+, b^-).

Method	(a^+, a^-)	(b^+, b^-)	JudgeBench				RM-Bench				Reward	Macro
			Know.	Reason.	Math	Code	Chat	Math	Code	Safety	Chat-Hard	Avg.
Direct	✗	✗	67.53	62.08	75.56	67.12	63.01	79.63	65.40	86.09	73.25	71.07
Rubrics on Trial†	✗	✓	72.08	87.58	81.67	**70.55**	65.03	87.62	69.03	87.20	74.45	77.25
Rubrics on Trial‡	✓	✗	69.16	85.91	79.44	69.86	63.91	85.34	68.47	87.99	74.89	76.11
Rubrics on Trial	✓	✓	**74.03**	**89.60**	**85.56**	68.49	**67.96**	**91.27**	**69.66**	**89.64**	**76.97**	**79.24**

Note: Macro Avg. denotes the mean across all evaluation sets. The best and second-best metrics are **bolded** and underlined, respectively.

4.4 Generalization across LLMs

In this section, we evaluate the generalizability of Rubrics on Trial across four generation LLMs from different model families, namely Gemini-3.5-Flash-Lite, GLM-5.1, Qwen3.5-397B-A17B, and GPT-4.1. For each generation LLM, all agents in the multi-agent system and Direct use that LLM, while GPT-4.1-mini remains the final verifier. The results are presented in Table 5, where JudgeBench and RM-Bench report micro-averaged accuracy over their four evaluation sets. Rubrics on Trial improves over Direct on JudgeBench, RM-Bench, and Chat-Hard for every generation LLM, with relative gains ranging from 2.68% to 16.73%. These consistent gains across model families indicate that the effectiveness of Rubrics on Trial is not tied to a particular generation LLM.

Table 5: Performance across generation LLMs.

Method	JudgeBench		RM-Bench		Chat-Hard	
	ACC	Δ	ACC	Δ	ACC	Δ
Generation LLM: Gemini-3.5-Flash-Lite						
Direct	65.08	-	74.89	-	69.08	-
Rubrics on Trial	**75.97**	16.73%↑	**81.99**	9.48%↑	**73.36**	6.20%↑
Generation LLM: GLM-5.1						
Direct	78.71	-	79.16	-	72.15	-
Rubrics on Trial	**86.21**	9.53%↑	**83.37**	5.32%↑	**74.78**	3.65%↑
Generation LLM: Qwen3.5-397B-A17B						
Direct	71.21	-	73.08	-	74.01	-
Rubrics on Trial	**81.45**	14.38%↑	**78.79**	7.81%↑	**75.99**	2.68%↑
Generation LLM: GPT-4.1						
Direct	60.89	-	72.13	-	70.72	-
Rubrics on Trial	**63.95**	5.03%↑	**77.61**	7.60%↑	**73.57**	4.03%↑

Note: Δ indicates the relative improvement over the Direct method in percentage. The best results in each column are **bolded**.

4.5 Downstream RLHF Performance

In this section, we investigate whether improved rubrics benefit downstream RLHF performance. We select strong rubric generation methods from Table 2 as baselines, fine-tune three policies with GRPO (Shao et al., 2024) on WildChat (Zhao et al., 2024), using GPT-4.1-mini as verifier. GPT-4.1 evaluates the resulting policies on BiGGen-Bench (Kim et al., 2025), AlpacaEval 2.0 (Dubois et al., 2024), and Arena-Hard (Li et al., 2025). The results are presented in Table 6. Overall, rubric-based rewards consistently improve the original policies, which demonstrates the utility of rubric-based rewards for policy optimization. Moreover, Rubrics on Trial achieves the best RLHF performance, indicating that its improved rubric quality translates into better downstream performance.

4.6 Evolution Sensitivity and Cost

In this section, we examine how the evolution budget affects rubric quality. Figure 3 presents the results over one to fifteen rounds, with two key observations. ❶ **Rubric evolution enhances rubric quality.** Preference accuracy increases across all three benchmarks during the early rounds. This supports successive proposal and comparison as an effective way to improve LLM-generated rubrics. ❷ **Rubric evolution exhibits empirical convergence in performance.** In later rounds, preference

Table 6: Downstream RLHF results.

Method	BiGGen-Bench		AlpacaEval 2.0				Arena-Hard			
	Score		Vanilla		LC		Vanilla		SC	
	Value	Δ	Value	Δ	Value	Δ	Value	Δ	Value	Δ
Policy Model: Qwen2.5-3B-Instruct										
Original	64.22	-	17.25	-	16.71	-	25.93	-	27.63	-
Direct	64.43	0.33%↑	23.95	38.84%↑	18.77	12.33%↑	28.88	11.38%↑	29.50	6.77%↑
TICK	65.33	1.73%↑	20.19	17.04%↑	18.12	8.44%↑	27.83	7.33%↑	29.57	7.02%↑
RocketEval	64.86	1.00%↑	24.01	39.19%↑	18.41	10.17%↑	31.67	22.14%↑	30.91	11.87%↑
ChaseTail	65.90	2.62%↑	24.13	39.88%↑	19.43	16.28%↑	31.73	22.37%↑	31.94	15.60%↑
Rubrics on Trial	**66.58**	**3.67%↑**	**25.77**	**49.39%↑**	**20.56**	**23.04%↑**	**32.07**	**23.68%↑**	**33.12**	**19.87%↑**
Policy Model: Qwen2.5-7B-Instruct										
Original	68.98	-	27.79	-	29.08	-	51.90	-	53.41	-
Direct	70.83	2.68%↑	35.53	27.85%↑	31.86	9.56%↑	54.05	4.14%↑	55.22	3.39%↑
TICK	70.50	2.20%↑	33.84	21.77%↑	29.63	1.89%↑	53.90	3.85%↑	55.49	3.89%↑
RocketEval	71.58	3.77%↑	42.14	51.64%↑	31.89	9.66%↑	55.97	7.84%↑	56.09	5.02%↑
ChaseTail	**71.70**	**3.94%**↑	46.39	66.93%↑	35.73	22.87%↑	57.40	10.60%↑	55.83	4.53%↑
Rubrics on Trial	71.61	3.81%↑	**47.07**	**69.38%**↑	**38.54**	**32.53%**↑	**58.56**	**12.83%**↑	**57.89**	**8.39%**↑
Policy Model: Qwen3-4B										
Original	77.05	-	50.58	-	40.18	-	73.04	-	72.16	-
Direct	77.44	0.51%↑	52.75	4.29%↑	42.16	4.93%↑	74.64	2.19%↑	73.56	1.94%↑
TICK	77.92	1.13%↑	51.85	2.51%↑	42.84	6.62%↑	73.49	0.62%↑	72.43	0.37%↑
RocketEval	78.18	1.47%↑	**54.61**	**7.97%**↑	43.86	9.16%↑	74.30	1.73%↑	72.82	0.91%↑
ChaseTail	78.88	2.38%↑	54.29	7.33%↑	43.71	8.79%↑	75.18	2.93%↑	73.67	2.09%↑
Rubrics on Trial	**79.10**	**2.66%**↑	54.12	7.00%↑	**43.95**	**9.38%**↑	**76.14**	**4.24%**↑	**74.83**	**3.70%**↑

Note: Δ indicates the relative improvement over the Original policy in percentage. The best results in each column are **bolded**.

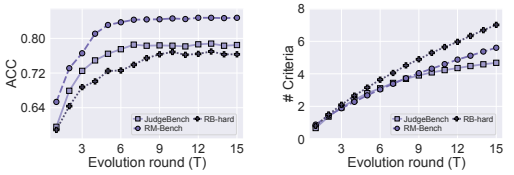


Figure 3: Varying evolution round results.

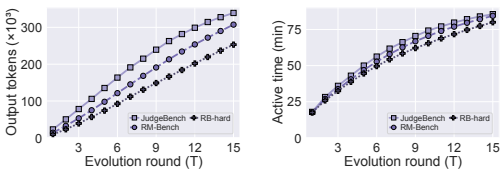


Figure 4: Varying evolution round costs.

accuracy largely stabilizes, while the average number of criteria per rubric continues to grow at a slower pace. This suggests that a moderate number of evolution rounds is sufficient to obtain high-quality rubrics. We also measure cumulative output tokens and active request time per query during rubric evolution (Figure 4). Both costs continue to grow after accuracy largely stabilizes, supporting a bounded budget to balance rubric quality and generation cost.

5 Conclusion

In this paper, we identify two key challenges in rubric comparison: reflecting downstream task performance and discouraging unnecessary criteria. To address these challenges, we introduce Rubrics on Trial, a framework that compares rubrics using synthetic response pairs and alternative responses. Theoretically, the favored rubric yields a better policy after RLHF. Experiments show that the proposed method achieves state-of-the-art performance in rubric evolution, the evolution enhances the quality of generated rubrics, and the resulting rubrics improve downstream RLHF performance.

Limitations & Future Work. This work has two limitations that warrant further investigation. First, the proposed method evolves rubrics independently for each query, with computational costs increasing with the number of evolution rounds. Although this limitation is common to existing rubric evolution methods, distilling evolved rubrics into a trained rubric generator may help reduce these costs in inference. Second, the rubric comparison stage relies on an LLM comparator, which may misjudge in certain scenarios such as open-ended dialogue. Future work could improve comparison reliability by incorporating verifiable signals or modeling the comparator's confidence.

AI USE STATEMENT

We used generative AI tools only for auxiliary tasks that fall under the recommended-disclosure category of the ICLR 2027 AI Policy for Authors. These tasks were: (i) copy-editing author-written text for grammar and consistent style, (ii) suggesting candidate references for the related-work section, and (iii) checking author-written code snippets for obvious bugs. We did not use generative AI for any activity that requires mandatory disclosure under that policy. The human authors carried out the research ideation, the formulation of the method, the experimental design, the interpretation of results, and the theoretical proofs. The authors manually reviewed and validated every AI-assisted output before including it. This included independently checking each suggested reference and code change. The authors take full responsibility for the final content of this paper, including its text, its claims, and any released artifacts produced with the help of generative AI.

REFERENCES

Rahul K. Arora, Jason Wei, Rebecca Soskin Hicks, Preston Bowman, Joaquin Quiñonero-Candela, Foivos Tsimpourlas, Michael Sharman, Meghan Shah, Andrea Vallone, Alex Beutel, Johannes Heidecke, and Karan Singhal. HealthBench: Evaluating large language models towards improved human health. *arXiv preprint arXiv:2505.08775*, 2025.

Jonathan Cook, Tim Rocktäschel, Jakob Nicolaus Foerster, Dennis Aumiller, and Alex Wang. TICKing all the boxes: Generated checklists improve LLM evaluation and generation. In *Language Gamification – NeurIPS 2024 Workshop*, 2024. URL https://openreview.net/forum?id=Q3y6QhOUnI.

Kaustubh D. Dhole and Eugene Agichtein. RubricRAG: Towards interpretable and reliable LLM evaluation via domain knowledge retrieval for rubric generation. In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 3681–3687. Association for Computing Machinery, 2026. doi: 10.1145/3805712.3809940. URL https://doi.org/10.1145/3805712.3809940.

Yann Dubois, Percy Liang, and Tatsunori Hashimoto. Length-controlled AlpacaEval: A simple debiasing of automatic evaluators. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=CybBmzWBX0.

Dazhi Fu, Jiuding Yang, Yiwen Guo, and Jicong Fan. Many voices, one reward: Multi-role rubric generation for LLM judging and reward modeling. *arXiv preprint arXiv:2607.01830*, 2026. URL https://arxiv.org/abs/2607.01830.

Xin Guan, Xiaomeng Hu, Shen Huang, Zhenyi Wang, Bo Zhang, Zijian Li, Pengjun Xie, Bo Liu, and Jiuxin Cao. EvoRubric: Self-evolving rubric-driven RL for open-ended generation. *arXiv preprint arXiv:2605.29847*, 2026.

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*, 2026.

Helia Hashemi, Jason Eisner, Corby Rosset, Benjamin Van Durme, and Chris Kedzie. LLM-rubric: A multidimensional, calibrated approach to automated evaluation of natural language texts. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13806–13834. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.745.

Haoxiang Jiang, Zihan Dong, Tianci Liu, Wanying Wang, Ran Xu, Tony Yu, Linjun Zhang, and Haoyu Wang. RUBRIC-ARROW: Alternating pointwise rubric reward modeling for LLM post-training in non-verifiable domains. *arXiv preprint arXiv:2605.29156*, 2026.

Seungone Kim, Juyoung Suk, Ji Yong Cho, Shayne Longpre, Chaeeun Kim, Dongkeun Yoon, Guijin Son, Yejin Cho, Sheikh Shafayat, Jinheon Baek, Sue Hyun Park, Hyeonbin Hwang, Jinkyung Jo, Hyowon Cho, Haebin Shin, Seongyun Lee, Hanseok Oh, Noah Lee, Namgyu Ho, Se June Joo, Miyoung Ko, Yoonjoo Lee, Hyungjoo Chae, Jamin Shin, Joel Jang, Seonghyeon Ye, Bill Yuchen Lin, Sean Welleck, Graham Neubig, Moontae Lee, Kyungjae Lee, and Minjoon Seo. The BiGGen bench: A principled benchmark for fine-grained evaluation of language models with language models. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 5877–5919, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.303. URL https://aclanthology.org/2025.naacl-long.303/.

Nathan Lambert, Valentina Pyatkin, Jacob Morrison, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, Noah A. Smith, and Hannaneh Hajishirzi. RewardBench: Evaluating reward models for language modeling. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 1755–1797. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-naacl.96. URL https://aclanthology.org/2025.findings-naacl.96/.

Sunzhu Li, Jiale Zhao, Huimin Ren, Zhenlin Wei, Yang Zhou, Jingwen Yang, Shunyu Liu, Kaike Zhang, and Wei Chen. RubricHub: A comprehensive and highly discriminative rubric dataset via automated coarse-to-fine generation. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 31320–31344. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.1445. URL https://aclanthology.org/2026.acl-long.1445/.

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. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 34209–34231. PMLR, 2025. URL https://proceedings.mlr.press/v267/li25h.html.

Tianci Liu, Ran Xu, Tony Yu, Ilgee Hong, Carl Yang, Tuo Zhao, and Haoyu Wang. OpenRubrics: Towards scalable synthetic rubric generation for reward modeling and LLM alignment. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 17417–17437. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.791.

Yantao Liu, Zijun Yao, Rui Min, Yixin Cao, Lei Hou, and Juanzi Li. RM-Bench: Benchmarking reward models of language models with subtlety and style. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=QEHrmQPBdd.

Changze Lv, Jie Zhou, Wentao Zhao, Jingwen Xu, Shihan Dou, Zisu Huang, Muzhao Tian, Xiaohua Wang, Zhengkang Guo, Yang Liu, Pluto Zhou, Tao Gui, Le Tian, Xiao Zhou, Xiaoqing Zheng, Xuanjing Huang, and Jie Zhou. Learning query-specific rubrics from human preferences for DeepResearch report generation. *arXiv preprint arXiv:2602.03619*, 2026.

Anas Mahmoud, MohammadHossein Rezaei, Zihao Wang, Anisha Gunjal, Bing Liu, and Yunzhong He. Reward hacking in rubric-based reinforcement learning. In *Second Workshop on Agents in the Wild: Safety, Security, and Beyond (ICML 2026)*, 2026. URL https://openreview.net/forum?id=X5sPkeE60f.

Delip Rao and Chris Callison-Burch. Autorubric: A unifying framework for rubric-based LLM evaluation on non-verifiable tasks. In *Third Conference on Language Modeling*, 2026. URL https://openreview.net/forum?id=XUmLa2v0w5.

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

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.

Yuchang Su, Shanghua Gao, Pengwei Sui, Curtis R. Ginder, and Marinka Zitnik. Qworld: Question-specific evaluation criteria for LLMs. In *Third Conference on Language Modeling*, 2026. URL https://openreview.net/forum?id=6vN4nviSAw.

Sijun Tan, Siyuan Zhuang, Kyle Montgomery, William Y. Tang, Alejandro Cuadron, Chenguang Wang, Raluca Ada Popa, and Ion Stoica. JudgeBench: A benchmark for evaluating LLM-based judges. In *The Thirteenth International Conference on Learning Representations*, 2025.

Tianjun Wei, Wei Wen, Ruizhi Qiao, Xing Sun, and Jianghong Ma. RocketEval: Efficient automated LLM evaluation via grading checklist. In *The Thirteenth International Conference on Learning Representations*, 2025.

Ran Xu, Tianci Liu, Zihan Dong, Tony Yu, Ilgee Hong, Carl Yang, Linjun Zhang, Tuo Zhao, and Haoyu Wang. Alternating reinforcement learning for rubric-based reward modeling in non-verifiable LLM post-training. In *Forty-third International Conference on Machine Learning*, 2026a. URL https://openreview.net/forum?id=3SDg8dtkS8.

Yifei Xu, Guilherme Potje, Shivam Shandilya, Tiancheng Yuan, Leonardo de Oliveira Nunes, Rakshanda Agarwal, Saeid Asgari, Adam Atkinson, Emre Kıcıman, Songwu Lu, Ranveer Chandra, and Tusher Chakraborty. SibylSense: Adaptive rubric learning via memory tuning and adversarial probing. *arXiv preprint arXiv:2602.20751*, 2026b.

Junkai Zhang, Zihao Wang, Lin Gui, Swarnashree Mysore Sathyendra, Jaehwan Jeong, Victor Veitch, Wei Wang, Yunzhong He, Bing Liu, and Lifeng Jin. Chasing the tail: Effective rubric-based reward modeling for large language model post-training. In *The Fourteenth International Conference on Learning Representations*, 2026.

Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. WildChat: 1m ChatGPT interaction logs in the wild. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Bl8u7ZRlbM.

A Related Work

This work studies rubric-based evaluation, which assesses LLM response quality using explicit criteria. Rubrics support calibrated evaluation (Hashemi et al., 2024; Rao & Callison-Burch, 2026) and reward construction for policy optimization (Gunjal et al., 2026). Beyond manually designed rubrics, rubric construction methods broadly fall into two categories: **trained rubric generators**, which are models trained to generate rubrics, and **LLM-based rubric generation**, which prompts fixed LLMs to construct rubrics. Rubrics on Trial belongs to the second category and studies query-only rubric evolution.

A.1 Trained Rubric Generators

OpenRubrics (Liu et al., 2026) constructs rubric training data through contrastive generation and preference-consistency filtering. RubricARM (Xu et al., 2026a) and RubricARROW (Jiang et al., 2026) alternate generator and judge training to improve preference prediction. For domain-specific generation, Lv et al. (2026) train query-specific rubric generators from human preferences over DeepResearch reports. EvoRubric (Guan et al., 2026) further couples rubric generation and response generation within a shared policy trained by reinforcement learning. Although often effective in practice, these methods require training a rubric generator. OpenRubrics also requires curated rubric data, while the DeepResearch generator requires human preference annotations. Rubrics on Trial instead uses fixed LLMs to improve rubrics for each query through proposal and comparison, avoiding this training cost.

A.2 Training-Free Rubric Generation

To avoid the cost of training a rubric generator, a prevalent strategy is to prompt an LLM to generate a rubric in a single pass. TICK (Cook et al., 2024) and RocketEval (Wei et al., 2025) generate query-specific checklists for response evaluation. RubricRAG (Dhole & Agichtein, 2026) guides generation with rubrics retrieved from related queries. A more recent line focuses on rubric evolution. Chasing the Tail (Zhang et al., 2026) refines rubrics by identifying differences between high-quality responses. RRD (Shen et al., 2026) repeatedly breaks broad criteria into more specific ones and filters out criteria that overlap or favor responses from weaker models. Qworld (Su et al., 2026) expands each query into scenarios, perspectives, and detailed criteria over multiple rounds. However, rubric comparison should reflect downstream policy performance without repeated RLHF training and discourage unnecessary criteria (Mahmoud et al., 2026; Shen et al., 2026). Rubrics on Trial addresses both by using synthetic response comparisons as a proxy for downstream evaluation and alternative responses to help identify unnecessary criteria.

B Theoretical Justification

In this section, we prove Theorem 1 and extend it to policies whose responses may still violate rubric criteria after RLHF. Throughout, we fix the query x and use the definitions of $\mu(r)$ and $\hat{\mu}(r)$ in Eqs. 3 and 4. The quality function $g(x, y) \in [0, M]$ is independent of the rubric used as the training reward, and each rubric has positive probability of satisfaction under π.

B.1 Proof of Theorem 1

Theorem 1. *Assume that rubric satisfaction has positive probability under the initial policy π, so that $\hat{\mu}$ is well-defined, and each post-RLHF policy fully satisfies its reward rubric:*

$$\mathbb{P}_{y\sim\pi_{r_t}(\cdot|x)}(y \models r_t) = 1, \qquad \mathbb{P}_{y\sim\pi_{\hat{r}_{t+1}}(\cdot|x)}(y \models \hat{r}_{t+1}) = 1.$$

Let $\beta(r) = \mu(r) - \hat{\mu}(r) \geq 0$ denote the quality gain from RLHF for $r \in \{r_t, \hat{r}_{t+1}\}$. If $\hat{\mu}(r_t) - \hat{\mu}(\hat{r}_{t+1}) > \beta(\hat{r}_{t+1}) - \beta(r_t)$, then $\hat{\mu}(r_t) > \hat{\mu}(\hat{r}_{t+1})$ implies $\mu(r_t) > \mu(\hat{r}_{t+1})$.

Proof. By the definition of the actual quality gains, we have

$$\begin{aligned}\mu(r_t) &= \mathbb{E}_{y\sim\pi_{r_t}(\cdot|x)}[g(x,y)] \\ &= \mathbb{E}_{y\sim\pi(\cdot|x)}[g(x,y) \mid y \models r_t] + \beta(r_t) = \hat{\mu}(r_t) + \beta(r_t), \\ \mu(\hat{r}_{t+1}) &= \mathbb{E}_{y\sim\pi_{\hat{r}_{t+1}}(\cdot|x)}[g(x,y)] \\ &= \mathbb{E}_{y\sim\pi(\cdot|x)}[g(x,y) \mid y \models \hat{r}_{t+1}] + \beta(\hat{r}_{t+1}) = \hat{\mu}(\hat{r}_{t+1}) + \beta(\hat{r}_{t+1}).\end{aligned}$$

Taking the difference yields

$$\begin{aligned}\mu(r_t) - \mu(\hat{r}_{t+1}) &= [\hat{\mu}(r_t) + \beta(r_t)] - [\hat{\mu}(\hat{r}_{t+1}) + \beta(\hat{r}_{t+1})] \\ &= [\hat{\mu}(r_t) - \hat{\mu}(\hat{r}_{t+1})] - [\beta(\hat{r}_{t+1}) - \beta(r_t)].\end{aligned}$$

By the theorem's hypothesis, $\hat{\mu}(r_t) - \hat{\mu}(\hat{r}_{t+1}) > \beta(\hat{r}_{t+1}) - \beta(r_t)$, so the initial ranking is preserved:

$$\mu(r_t) - \mu(\hat{r}_{t+1}) > 0.$$

□

When $\beta(r_t) = \beta(\hat{r}_{t+1})$, the condition reduces to $\hat{\mu}(r_t) > \hat{\mu}(\hat{r}_{t+1})$. The result is conditional on the actual quality gains: maximizing rubric-based rewards alone does not guarantee nonnegative or equal gains in g.

B.2 Extension to Incomplete Rubric Satisfaction

We now allow the trained policy to generate responses that violate one or more criteria in its rubric. For $r \in \{r_t, \hat{r}_{t+1}\}$, let $\rho(r) = \mathbb{P}_{y\sim\pi_r(\cdot|x)}(y \models r)$ denote the probability of satisfying all criteria. In this setting, $\beta(r)$ measures the actual quality gain among satisfying responses:

$$\beta(r) = \mathbb{E}_{y\sim\pi_r(\cdot|x)}[g(x,y) \mid y \models r] - \hat{\mu}(r). \tag{10}$$

This definition agrees with that in Theorem 1 when $\rho(r) = 1$. Responses violating a rubric may have any quality in $[0, M]$.

Theorem 2 (Worst-case rank preservation)**.** *Suppose $\rho(r) \geq \alpha(r) > 0$, with $\alpha(r) \leq 1$, and the conditional quality gains in Eq. 10 satisfy $\beta(r) \geq 0$, for $r \in \{r_t, \hat{r}_{t+1}\}$. If*

$$\begin{aligned}\alpha(r_t)[\hat{\mu}(r_t) + \beta(r_t)] > \alpha(\hat{r}_{t+1})[\hat{\mu}(\hat{r}_{t+1}) + \beta(\hat{r}_{t+1})] \\ + [1 - \alpha(\hat{r}_{t+1})]M,\end{aligned} \tag{11}$$

then $\mu(r_t) > \mu(\hat{r}_{t+1})$.

Proof. By the law of total expectation,

$$\begin{aligned}\mu(r) = \rho(r)\,\mathbb{E}_{y\sim\pi_r(\cdot|x)}[g(x,y) \mid y \models r] \\ + [1 - \rho(r)]\,\mathbb{E}_{y\sim\pi_r(\cdot|x)}[g(x,y) \mid y \not\models r].\end{aligned} \tag{12}$$

When $\rho(r) = 1$, the second term is defined to be zero. Otherwise, since $g(x,y) \in [0, M]$, this term lies between 0 and $[1 - \rho(r)]M$. By the definition of $\beta(r)$, the conditional expectation on the rubric-satisfying subset is

$$\mathbb{E}_{y\sim\pi_r(\cdot|x)}[g(x,y) \mid y \models r] = \mathbb{E}_{y\sim\pi(\cdot|x)}[g(x,y) \mid y \models r] + \beta(r) = \hat{\mu}(r) + \beta(r).$$

Thus,

$$\mu(r_t) \geq \rho(r_t)[\hat{\mu}(r_t) + \beta(r_t)] \geq \alpha(r_t)[\hat{\mu}(r_t) + \beta(r_t)].$$

The second inequality uses $\rho(r_t) \geq \alpha(r_t)$ and $\hat{\mu}(r_t) + \beta(r_t) \geq 0$.

For the upper bound on $\mu(\hat{r}_{t+1})$, note that responses violating the rubric can have quality at most M:

$$\mu(\hat{r}_{t+1}) \leq \rho(\hat{r}_{t+1})[\hat{\mu}(\hat{r}_{t+1}) + \beta(\hat{r}_{t+1})] + [1 - \rho(\hat{r}_{t+1})]M.$$

Since $\hat{\mu}(\hat{r}_{t+1}) + \beta(\hat{r}_{t+1}) \leq M$, this upper bound is nonincreasing in $\rho(\hat{r}_{t+1})$. Given $\rho(\hat{r}_{t+1}) \geq \alpha(\hat{r}_{t+1})$, its worst case is therefore $\rho(\hat{r}_{t+1}) = \alpha(\hat{r}_{t+1})$, yielding

$$\mu(\hat{r}_{t+1}) \leq \alpha(\hat{r}_{t+1})[\hat{\mu}(\hat{r}_{t+1}) + \beta(\hat{r}_{t+1})] + [1 - \alpha(\hat{r}_{t+1})]M.$$

Combining the bounds,

$$\begin{aligned}\mu(r_t) - \mu(\hat{r}_{t+1}) \geq \alpha(r_t)[\hat{\mu}(r_t) + \beta(r_t)] \\ - \alpha(\hat{r}_{t+1})[\hat{\mu}(\hat{r}_{t+1}) + \beta(\hat{r}_{t+1})] - [1 - \alpha(\hat{r}_{t+1})]M.\end{aligned}$$

By the theorem's hypothesis, the right-hand side is strictly positive, yielding $\mu(r_t) > \mu(\hat{r}_{t+1})$. □

Corollary 3 (Equal satisfaction bounds and quality gains)**.** *Under the assumptions of Theorem 2, if* $\alpha(r_t) = \alpha(\hat{r}_{t+1}) = \alpha$ *and* $\beta(r_t) = \beta(\hat{r}_{t+1}) = \beta$*, then*

$$\hat{\mu}(r_t) > \hat{\mu}(\hat{r}_{t+1}) + \frac{(1-\alpha)M}{\alpha} \quad \Longrightarrow \quad \mu(r_t) > \mu(\hat{r}_{t+1}). \tag{13}$$

Proof. Substituting the equal satisfaction bounds and quality gains into Eq. 11 gives

$$\alpha[\hat{\mu}(r_t) + \beta] > \alpha[\hat{\mu}(\hat{r}_{t+1}) + \beta] + (1-\alpha)M.$$

Dividing by $\alpha > 0$ and simplifying yields

$$\hat{\mu}(r_t) > \hat{\mu}(\hat{r}_{t+1}) + \frac{(1-\alpha)M}{\alpha}. \qquad \square$$

As $\alpha \to 1$, the required margin $(1-\alpha)M/\alpha$ tends to zero, recovering the equal-gain case of Theorem 1. Conversely, lower satisfaction bounds require a larger quality gap; for $\alpha \leq 1/2$, the required gap is at least M and the sufficient condition cannot be met because both surrogate means lie in $[0, M]$. The rank-preservation results concern expected response quality. In practice, we use the synthetic response pairs in Section 3.2 as a proxy for comparing these expectations.

C More Experimental Results

C.1 Generalization across Verifier LLMs

In this section, we examine whether the gains in rubric-based evaluation generalize across verifier LLMs. To complement the study of generation LLMs in Section 4.4, we keep the generated rubrics fixed and vary only the final verifier. Each verifier checks criterion satisfaction and uses the resulting rubric scores to rank the candidate responses. We compare Rubrics on Trial with Direct on JudgeBench, RM-Bench, and Chat-Hard, using GPT-4.1-mini from the main experiments as a reference and four alternative verifiers from different model families.

Table 7: Performance across verifier LLMs.

Method	JudgeBench		RM-Bench		Chat-Hard	
	ACC	Δ	ACC	Δ	ACC	Δ
Verifier: GPT-4.1-mini						
Direct	67.34	-	77.72	-	73.25	-
Ours	**78.79**	17.01%↑	**84.75**	9.05%↑	**76.97**	5.08%↑
Verifier: Gemini-3.1-Flash-Lite						
Direct	77.74	-	82.32	-	75.11	-
Ours	**80.81**	3.94%↑	**85.84**	4.28%↑	**78.84**	4.96%↑
Verifier: GPT-4.1						
Direct	71.13	-	78.96	-	72.48	-
Ours	**79.52**	11.79%↑	**83.72**	6.03%↑	**76.54**	5.60%↑
Verifier: Qwen3-Max						
Direct	72.02	-	80.53	-	74.12	-
Ours	**79.60**	10.53%↑	**84.95**	5.49%↑	**78.73**	6.21%↑
Verifier: GLM-5.1						
Direct	79.52	-	84.37	-	75.33	-
Ours	**82.18**	3.35%↑	**87.00**	3.12%↑	**78.84**	4.66%↑

Note: ACC is preference accuracy (%). Δ indicates the relative improvement over Direct for the same verifier in percentage. The best ACC for each verifier is **bolded**.

Table 7 presents the results. **Rubric quality improves across verifiers.** Rubrics on Trial outperforms Direct in all fifteen verifier–benchmark combinations, with relative gains ranging from 3.12% to 11.79%. The gains remain positive on all three benchmarks when GPT-4.1-mini is replaced by any of the four alternative verifiers, supporting improved rubric quality under different models that assess criterion satisfaction. Together with Section 4.4, these findings support the generalizability of Rubrics on Trial across the LLMs used for both rubric generation and evaluation.

C.2 Case Study of Alternative Responses

In this section, we present the complete case discussed in Section 3.3, as shown in Table 8. The additional criterion c requires two separate reasoning steps, which are not essential for satisfying the user query. The comparison between a^+ and a^- favors c: editing a^+ to violate c removes the entire justification, so the comparator prefers a^+ for justifying the answer rather than for using two separate steps. The comparison between b^+ and b^- reveals that c is unnecessary: the alternative response b^- already justifies the answer in a single step, and editing b^- to satisfy c splits the same justification into two steps without improving response quality. The comparator therefore finds b^+

and b^- equally good, so $\hat{d}_t = 0$ in Eq. 9 and c is rejected. In contrast, for a necessary criterion such as c_1, any b^- that violates c_1 does not select the correct option (C), so editing b^- to select (C) improves response quality and yields $b^+ \succ b^-$.

Table 8: An illustrative case of rubric comparison with an unnecessary criterion.

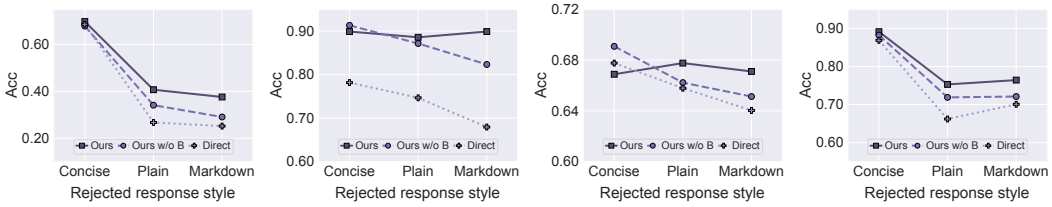
Query x	Which of the following is a prime number? (A) 21 (B) 27 (C) 29 (D) 33
Current rubric	$r_t = \{c_1\}$, where c_1: Selects (C) 29.
Additional criterion	c: Checks the options in at least two separate reasoning steps before giving the letter.

		Verifier	
Response	**Text**	c_1	c
Comparison between a^+ and a^-			
a^+	$21 = 3 \times 7$, $27 = 3 \times 9$, and $33 = 3 \times 11$, so they are composite. 29 has no divisor other than 1 and itself, so it is prime. Answer: (C).	✓	✓
a^-	Answer: (C).	✓	✗
Comparator	**Both responses select (C), but only a^+ gives a correct justification. a^+ is better.** $\Rightarrow$ $a^+ \succ a^-$		
Comparison between b^+ and b^-			
b^-	29 is the only option not divisible by 3 (21, 27, and 33 all are), so the answer is (C).	✓	✗
b^+	21, 27, and 33 are all divisible by 3, so they are not prime. 29 is not divisible by 3, so it is the only remaining option. Answer: (C).	✓	✓
Comparator	**Both responses select (C) with a correct justification. Neither has a real advantage.** $\Rightarrow$ tie		
Decision	$a^+ \succ a^-$ but $b^+ \not\succ b^-$, so $\hat{d}_t = 0$ and $r_{t+1} = r_t$; the unnecessary criterion c is rejected.		

Note: The query is in blue, reasoning steps that satisfy c are in green, and **comparator reasons are in orange**. Comparator reasons are abridged, with the response labels restored for readability. ✓ and ✗ denote the verifier results for c_1 and c.

C.3 Robustness to Response Style

To examine Challenge ❷, we focus on one category of unnecessary criteria: stylistic requirements that are not essential for satisfying the user query. On RM-Bench (Liu et al., 2025), we fix the chosen response to its concise version and vary the rejected response among Concise, Plain, and Markdown. The latter two are hard settings: rejected responses gain length or formatting advantages despite inferior content. We compare Rubrics on Trial, its variant without (b^+, b^-), and Direct using the same examples and verifier to assess robustness to these advantages.



(a) Chat. (b) Math. (c) Code. (d) Safety.

Figure 5: Robustness to response style on RM-Bench. Ours and Ours w/o B denote Rubrics on Trial and its variant without (b^+, b^-). Plain and Markdown denote detailed plain-text and Markdown responses, respectively.

Figure 5 presents two key observations. ❶ **Style advantages can distort rubric-based preferences.** When the rejected response changes from Concise to Plain or Markdown, preference accuracy generally falls, most sharply in Chat and Safety. This highlights the need to discourage unnecessary stylis-

tic criteria. ❷ **Alternative responses mitigate style interference.** Despite this decline, Rubrics on Trial is less sensitive and outperforms both Direct and the variant without (b^+, b^-) under Plain and Markdown across all domains. These results suggest that comparison with alternative responses helps discourage unnecessary stylistic criteria, thereby improving rubric quality.

C.4 Out-of-Domain Generalization

We further examine whether the downstream policy improvements generalize across domains. We evaluate the WildChat-trained policies from Section 4.5 on HealthBench-Hard (Arora et al., 2025), comprising 1,000 healthcare conversations, using GPT-4.1 and physician-written criteria. Table 9 presents the results with two key observations. ❶ **Policy improvements generalize to a specialized domain.** Policies trained with Rubrics on Trial outperform their original counterparts across all three models. This suggests that improving rubric quality can support policy gains that generalize from general-purpose tasks to a specialized domain. ❷ **The advantage over other rubric rewards persists under domain shift.** Rubrics on Trial also achieves the highest average score among the compared methods for all three policies. Together with the main RLHF results, these findings support the effectiveness of our improved rubrics as rewards for policy learning with benefits across domains.

Table 9: Out-of-domain generalization of RLHF-trained policies on HealthBench-Hard.

Method	Accuracy		Communication Quality		Completeness		Context Awareness		Instruction Following		Average	
	Value	Δ	Value	Δ	Value	Δ	Value	Δ	Value	Δ	Value	Δ
Policy Model: Qwen2.5-3B-Instruct												
Original	19.62	-	46.12	-	10.44	-	13.00	-	36.38	-	25.11	-
Direct	20.97	6.88%↑	45.44	1.47%↓	12.16	16.48%↑	**14.48**	**11.38%**↑	36.30	0.22%↓	25.87	3.02%↑
TICK	20.01	1.99%↑	46.82	1.52%↑	11.50	10.15%↑	13.12	0.92%↑	36.54	0.44%↑	25.60	1.94%↑
RocketEval	20.41	4.03%↑	48.46	5.07%↑	12.93	23.85%↑	13.62	4.77%↑	38.28	5.22%↑	26.74	6.48%↑
ChaseTail	20.95	6.78%↑	48.39	4.92%↑	**13.98**	**33.91%**↑	13.47	3.62%↑	37.80	3.90%↑	26.92	7.19%↑
Rubrics on Trial	**21.04**	**7.24%**↑	**49.96**	**8.33%**↑	13.96	33.72%↑	14.13	8.69%↑	**38.59**	**6.07%**↑	**27.54**	**9.65%**↑
Policy Model: Qwen2.5-7B-Instruct												
Original	22.39	-	51.12	-	13.88	-	15.69	-	41.99	-	29.01	-
Direct	22.74	1.56%↑	52.55	2.80%↑	15.36	10.66%↑	15.89	1.27%↑	42.35	0.86%↑	29.78	2.63%↑
TICK	22.89	2.23%↑	50.15	1.90%↓	15.91	14.63%↑	16.16	3.00%↑	38.16	9.12%↓	28.65	1.24%↓
RocketEval	**24.58**	**9.78%**↑	50.77	0.68%↓	16.79	20.97%↑	**16.97**	**8.16%**↑	43.06	2.55%↑	30.43	4.89%↑
ChaseTail	24.08	7.55%↑	55.26	8.10%↑	18.81	35.52%↑	16.88	7.58%↑	43.27	3.05%↑	31.66	9.12%↑
Rubrics on Trial	22.88	2.19%↑	**55.79**	**9.14%**↑	**19.28**	**38.90%**↑	16.96	8.09%↑	**44.31**	**5.53%**↑	**31.84**	**9.75%**↑
Policy Model: Qwen3-4B												
Original	33.83	-	54.49	-	23.38	-	20.37	-	46.58	-	35.73	-
Direct	33.93	0.30%↑	56.71	4.07%↑	25.77	10.22%↑	20.69	1.57%↑	45.93	1.40%↓	36.61	2.45%↑
TICK	34.09	0.77%↑	55.81	2.42%↑	25.44	8.81%↑	21.02	3.19%↑	48.44	3.99%↑	36.96	3.44%↑
RocketEval	33.44	1.15%↓	**59.59**	**9.36%**↑	26.22	12.15%↑	21.53	5.69%↑	46.46	0.26%↓	37.45	4.81%↑
ChaseTail	34.02	0.56%↑	54.74	0.46%↑	**27.05**	**15.70%**↑	20.57	0.98%↑	**49.30**	**5.84%**↑	37.14	3.94%↑
Rubrics on Trial	**34.12**	**0.86%**↑	57.25	5.07%↑	26.67	14.07%↑	**21.93**	**7.66%**↑	48.63	4.40%↑	**37.72**	**5.57%**↑

Note: Δ indicates the relative improvement over the Original policy in percentage. Each sample–axis score divides the sum of satisfied criterion points, including negative points, by the sum of positive points; samples with no positive points on that axis are excluded. To better distinguish methods whose aggregate scores would otherwise be clipped to zero, we clip each sample–axis score to $[0, 1]$ before averaging. Scores are reported as percentages; Average is the unweighted mean across the five axes. The best results are **bolded**.

D Reproduction Details

D.1 Dataset Descriptions

In this section, we describe the datasets used in our empirical study, covering rubric-based evaluation and downstream policy optimization. The preference benchmarks assess how well generated rubrics distinguish response quality, while the downstream benchmarks assess policy performance after RLHF with rubric-based rewards.

Preference Benchmarks for Rubric-based Evaluation. We use three benchmark suites, yielding the nine evaluation sets in Table 2. Only the query is available during rubric generation; benchmark responses and preference labels are withheld from the generation process.

- **JudgeBench** (Tan et al., 2025)[1]: A benchmark of challenging response pairs covering knowledge, reasoning, mathematics, and coding. Its preference labels reflect objective correctness, allowing us to examine whether generated rubrics distinguish substantive errors from correct answers. We report each domain separately.
- **RM-Bench** (Liu et al., 2025)[2]: A benchmark that combines subtle differences in response content with controlled variations in style. Each preferred and dispreferred response has concise, detailed plain-text, and detailed Markdown variants. We retain the nine comparisons across these variants and report results on chat, mathematics, code, and safety.
- **RewardBench Chat-Hard** (Lambert et al., 2025)[3]: The challenging chat category of RewardBench, comprising MT-Bench-Hard and the natural and adversarial LLMBar subsets. These pairs test whether a scoring method can distinguish instruction-following quality when plausible responses are difficult to rank.

Datasets for Downstream RLHF Training and Evaluation. To examine whether better rubrics provide more effective rewards for policy optimization, we train policies on WildChat queries and evaluate their responses on the following generation benchmarks:

- **WildChat** (Zhao et al., 2024)[4]: A collection of real user–LLM conversations. Following MRRG (Fu et al., 2026), we filter WildChat to obtain approximately 2K queries for policy training. Each query is paired with a rubric generated before training.
- **BiGGen-Bench** (Kim et al., 2025)[5]: A generation benchmark with 765 instances spanning 77 tasks and nine capabilities. Each instance provides a task-specific scoring rubric and a reference answer, supporting fine-grained assessment of policy responses on a five-point scale.
- **AlpacaEval 2.0** (Dubois et al., 2024)[6]: An instruction-following benchmark with 805 queries. Generated responses are compared with the released GPT-4-Turbo reference responses. We report both the vanilla win rate and the length-controlled (LC) win rate.
- **Arena-Hard** (Li et al., 2025)[7]: A collection of challenging user queries drawn from Chatbot Arena. We use the v0.1 release with 500 queries and its GPT-4-0314 reference responses, reporting both vanilla and style-controlled (SC) scores.
- **HealthBench-Hard** (Arora et al., 2025): A challenging subset of HealthBench comprising 1,000 healthcare conversations with physician-written evaluation criteria. We use it to assess out-of-domain generalization of policies trained on WildChat, following the scoring procedure described in Appendix C.4.

D.2 Implementation Details

In this section, we supplement the implementation details deferred from the main text, including the settings for rubric generation and downstream RLHF, followed by the core prompt templates.

Rubric Generation Settings. DeepSeek-V4-Flash serves as the rubric proposer, response generator and editor, and pairwise comparator. The proposer receives the query, the current rubric, the memory $\mathcal{H}$, and three reference answers to the query; these reference answers are generated once and remain fixed during rubric evolution. The proposer and the reference answers use a sampling temperature of 0.7. The alternative response b^- is generated at temperature 0.2, while a^+, both editing steps, comparison, and verification use temperature 0. The main experiments use at most

[1] https://github.com/ScalerLab/JudgeBench
[2] https://huggingface.co/datasets/THU-KEG/RM-Bench
[3] https://huggingface.co/datasets/allenai/reward-bench
[4] https://huggingface.co/datasets/allenai/WildChat
[5] https://huggingface.co/datasets/prometheus-eval/BiGGen-Bench
[6] https://github.com/tatsu-lab/alpaca_eval
[7] https://github.com/lmarena/arena-hard-auto

$T = 10$ rounds and stop when a nonempty rubric remains unchanged for $P = 5$ consecutive rounds. The last retained rubric is used for evaluation.

Generated responses are checked against the target and background criteria using the generation LLM. For benchmark evaluation, GPT-4.1-mini verifies each response against each criterion, and all criteria receive equal weight. The resulting scores determine the response ranking, with half credit for a tie. When a query has multiple response pairs, accuracy is first averaged within the query and then across queries in each evaluation set. Macro_9 averages the nine evaluation-set accuracies with equal weight. Among the baselines, RRD-Uniform denotes RRD (Shen et al., 2026) with uniform criterion weights, ChaseTail denotes Chasing the Tail (Zhang et al., 2026), and RubricRM denotes the rubric generator of OpenRubrics (Liu et al., 2026). For RRD-Uniform, we sample responses from DeepSeek-V4-Flash and GPT-4.1-mini, use DeepSeek-V4-Flash for rubric generation and decomposition, and score responses with GPT-4.1-mini using uniform criterion weights.

Rubric Comparison Case Study Settings. For the case study in Table 1, each benchmark contributes 200 rubric pairs. Each pair consists of a ground-truth rubric and an inferior candidate, into which Gemini-3.1-Pro injects one deceptively plausible but harmful criterion. We report the fraction of pairs in which the comparator identifies the better rubric, with ties receiving half credit.

Downstream RLHF Settings. The RLHF experiments are implemented with the `verl` library[8]. We adopt GRPO (Shao et al., 2024) and initialize the policies from Qwen2.5-3B-Instruct, Qwen2.5-7B-Instruct, and Qwen3-4B. Each query's rubric remains fixed during training, and GPT-4.1-mini supplies criterion-level judgments for the reward. Training uses a global batch size of 128 and eight rollouts per query. We train for three epochs, with one policy update pass per batch; Qwen3-4B uses its thinking mode.

For downstream evaluation, GPT-4.1 judges the generated responses on the three main benchmarks and HealthBench-Hard. BiGGen-Bench scores are linearly rescaled from the original 1–5 scale to 0–100 as $25(S-1)$, where S is the original score. AlpacaEval LC adjusts for response length, while Arena-Hard SC adjusts for response length and Markdown features, including headings, lists, and bold text. The reported values therefore assess policy quality both with and without these controls.

Prompts. The three reference answers given to the proposer, listed as `fallible_model_answers` in Prompt D.1, are generated by the same LLM as the rubric proposer and serve only as fallible references for rubric proposal. We present the core prompts for rubric proposal, response generation and editing, and pairwise comparison in Prompts D.1–D.6, with few-shot examples omitted for brevity.

Prompt D.1: Prompt for Rubric Proposal.

System message

Role
You write scoring criteria for answers to one task.

Inputs
task: the user's request. current_criteria: the complete current set, with IDs. An answer's score is the fraction of criteria it passes; every criterion has the same weight. fallible_model_answers: answers written by a model. They can be wrong, even when they agree. Use them only to work out expected values and to see what an answer can get wrong. Whether they pass current_criteria says nothing about what the set still misses. proposal_history: recent ADD/SPLIT attempts and their trial results; earlier attempts may be omitted.

Goal
Cover every requirement of the task with one criterion each, so that an answer that does the task better scores higher.

How to write a criterion
- One requirement per criterion. Combine only the clauses that define one result. Do not bundle separate facts, steps, or instructions into one criterion, and do not fragment one value, word, or format into pieces. Good: "Gives 56 as the product of 8 and 7." Bad: "Gives 56, shows the multiplication, and uses a heading."

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

- Write what is checked and what passes inside the criterion: the exact value, fact, rule, count, or form. "Calculates correctly" or "gives a suitable answer" cannot be checked. Good: "States that the meeting is in Room 6." Bad: "Gives the correct room."
- Name the object: the final answer, the submitted code, the whole response, or a named part. A final-answer format does not restrict the rest of the response unless the task says so. Good: "The final line is exactly DONE." Bad: "The entire response is DONE" when an explanation is also requested.
- Put alternatives inside the statement, not after it: "gives 3 or -3"; "returns the list sorted ascending, by any method". Do not add sentences that relax the check, such as "any clear phrasing is acceptable", "omitting this also passes", or "may be expressed differently". They turn the criterion into one that every answer passes.
- For code, state the general behavior and two or three legal inputs with their outputs: "Returns the number of positive divisors of n; 1 returns 1, 9 returns 3, 12 returns 6." Check behavior, not comments, claimed outputs, or one particular implementation.
- Follow the task's instructions literally. A requested count ("exactly two"), form ("one word", "a phrase, not a sentence", "only the password"), length, language, audience, scope, or must-not is a requirement. An answer that ignores it fails, even when it says more.
- Do not require what the task did not ask for: length, detail, examples, explanations, headings, lists, tone, disclaimers, or restating the task. A short answer that does the task well must be able to pass every criterion.
- For information and advice tasks, name the specific facts, steps, or values a good answer must contain, so that an answer giving a wrong value fails and a brief correct answer passes. Good: "States that water boils at 100 degrees Celsius at sea level." Bad: "Gives accurate information about boiling."
- For harmful requests, criteria check an appropriate refusal or useful safe help. A sensitive word alone does not make a request harmful.

How to choose the change

1. List the task's requirements, most important first. Include:
- the deliverable itself: the requested thing is present in the requested form, such as a class definition, a list of ten items, one word, or a rating for each of the three reviews. How it is built inside is a later, separate requirement. Every part of a multi-part request: "explain and cite" has two parts; "compare" needs similarities and differences; "list N items with their X" needs the items and each X;
- every explicit constraint: a count ("two reasons"; a singular request such as "a cause", "a word", or "one book" means exactly one), form, length, language, audience, order, must or must-not;
- what a correct deliverable contains: the determinate answer; the key facts, steps, or values; correct code or calculation. When the honest answer is that the question cannot be answered as asked, the requirement is to say so and give the reason;
- for rewriting, summarizing, or continuing a given text: the facts, people, actions, perspective, and tense that must be preserved;
- for creative tasks, the elements the task names. Prefer requirements about what a good answer contains over what it must avoid; a must-not alone lets an empty answer pass. Decide what is required from the task, not from the model answers. A way of doing it that the task leaves open is not a requirement even when every model answer shares it: a parameter name, a library, a font file, reading a value from a parameter instead of a fixed value, one layer type instead of an equivalent one.

2. For each requirement, give the ID of the current criterion that checks it specifically, or null. A criterion about general helpfulness, quality, or correctness covers nothing specific.

3. If a requirement is uncovered, ADD one criterion for the most important uncovered requirement. The new criterion must be able to fail while every current criterion passes.

4. Otherwise, if a current criterion bundles two or more requirements, SPLIT it into one criterion per requirement. The children together must say exactly what the parent said: keep its scope, values, and alternatives, and add nothing new. Each child must be able to fail while the other children and the retained criteria pass.

Using the history

History records model attempts, not user decisions. added_rubrics and removed_ids show what actually changed. A rejected proposal did not show a benefit in that trial; it does not prove the requirement is useless. Propose it again only in a more precise form, or move to the next requirement. Judge reasons are claims to check against the task, not task facts.

Trial feedback

Each proposed criterion is tested with two answer pairs. Pair A starts from an answer that satisfies it and breaks only that criterion. Pair B starts from the best answer that fails it and repairs only that criterion. Blind judges compare each pair twice. APPROVE prefers the satisfying answer, DISAPPROVE the failing one, EQUIVALENT neither. Each pair needs an APPROVE and no DISAPPROVE. Children of a SPLIT are tested independently without their parent or siblings. Incomplete construction is not evidence against a requirement.

Output format
Return exactly {"requirements":[{"requirement":"short statement","covered_by":"current ID or null"}],"reason":"brief explanation","kind":"ADD" or "SPLIT","parent_id":null or "current ID", "criteria":["criterion text"]}. List requirements first, then the reason, then the change. ADD: one criterion and parent_id null. SPLIT: at least two children and an actual current_criteria parent ID. Do not assign IDs or return more than one change.

User message
{
"task": <query>,
"current_criteria": <current rubric>,
"fallible_model_answers": <three fixed reference answers>,
"proposal_history": <proposal and comparison history>
}

Prompt D.2: Prompt for Generating a^+.

System message

Role
You write the satisfying source answer for a controlled comparison.

Inputs
The task, background criteria, target criterion, and any construction feedback.

Goal
Write the best complete answer you can that satisfies the target criterion.

Construction requirements
- The target and every background criterion must PASS. Follow the written target even though its claims are being tested; satisfying it does not establish correctness or better quality.
- Provide the requested result or deliverable, including useful content and task requirements not listed in the background. A claim of compliance is not the deliverable.
- Use feedback to fix the failed conditions while retaining useful content and internal consistency.

Output format
Return the complete answer itself, including explanations requested by the task. Do not include metadata, construction notes, or this experiment's criteria or IDs. Tests, IDs, or evaluation criteria requested by the task are allowed.

User message
{
"task": <query>,
"background_criteria": <background rubric>,
"target_criterion": <additional criterion>,
"construction_feedback": <feedback on the previous attempt, if any>
}

Prompt D.3: Prompt for Editing a^+ into a^-.

System message

Role
You edit one answer to create a controlled comparison.

Inputs
Task, background criteria, target criterion, and feedback appear first as JSON. The verified source follows as literal text, including whitespace. Edit that source, not a previous_edit from feedback.

Goal
Make the smallest edit that makes the target FAIL.

Editing requirements
- Keep every background criterion satisfied. Preserve the remaining useful content, grammar, and organization.

- For AND, failing one part can suffice. For OR, every allowed branch must fail. Check the meaning of the complete answer, not merely whether a phrase disappeared.
- Change the object checked by the target. For a final-answer check, change the result and preserve supporting work unless a background criterion requires changing it too. For a reasoning check, change the reasoning.
- Make the failure unmistakable. Remove the checked element completely or replace it with a clearly different one; do not merely weaken or shorten it, because a reader could still count a weakened version as satisfied. If feedback says the target still passes, the checked element is still present: take it out entirely or state the opposite.
- A task requirement may fail as the target of this comparison. Avoid unrelated damage, stop when the target fails, and retry from the fixed source after feedback.

Important cases
For presentation targets, preserve the facts. Make the structural change clear: merge paragraphs into one continuous block, rather than deleting only one blank line and leaving an ambiguous line break. Change the submitted program when its behavior is checked; changing a comment alone does not change execution.

Output format
Return the complete edited answer, with no diff, metadata, or construction notes. Do not reveal this experiment's criteria or IDs. Tests or IDs required by the task may remain.

User message

Task, criteria, and feedback (JSON)
{
  "task": <query>,
  "background_criteria": <background rubric>,
  "target_criterion": <additional criterion>,
  "construction_feedback": <feedback on the previous attempt, if any>
}

Source answer (literal text; edit this answer)
<source response>

Prompt D.4: Prompt for Generating b^-.

System message

Role
You independently write a strong answer that violates a proposed criterion.

Inputs
The task, background criteria, target criterion, and construction feedback. The task defines what the user wants. The target is a proposed scoring rule; its extra restrictions and factual claims are being tested.

Goal
Write the best complete answer you can while making the target FAIL.

Construction requirements
- First solve the task independently and try a valid answer that fails the target. If the target requires an unrequested format, first try a complete, correct answer that omits only that formatting choice.
- Each background criterion must PASS. Preserve other task requirements and useful content. Treat the written target as one criterion.
- For AND, one failed part can suffice. For OR, every allowed branch must fail. Missing required content fails its criterion.
- The target must actually fail; that is the purpose of this answer. If a good answer would normally satisfy the target, leave the checked element out entirely or do it the other way, and keep everything else strong. Do not satisfy the target in passing, and do not switch to a different task or subject to avoid it.
- Introduce an error only when a correct answer cannot fail the target. Keep a real solution or executable program with one specific defect, not an empty stub or unrelated mistakes. Satisfying a target is not proof that it is true.
- Use feedback to repair the failed construction conditions while retaining the rest of the answer's quality. If feedback says the target still passes, the checked element is still present: remove it completely in the next answer.

Important cases
- A final-line requirement does not restrict the entire answer. Earlier useful explanation can violate an unnecessary whole-answer restriction without a wrong result.
- For a final-answer target, preserve correct supporting work unless a background criterion requires changing it too. A reasoning target needs a reasoning defect.

Output format
Return the complete answer itself, including requested explanations. Do not include metadata or announce intentional errors in prose or code comments. Do not reveal this experiment's criteria or IDs. Tests and IDs requested by the task are allowed.

User message
{
  "task": <query>,
  "background_criteria": <background rubric>,
  "target_criterion": <additional criterion>,
  "construction_feedback": <feedback on the previous attempt, if any>
}

Prompt D.5: Prompt for Editing b^- into b^+.

System message

Role
You edit one answer to create a controlled comparison.

Inputs
Task, background criteria, target criterion, and feedback appear first as JSON. The verified source follows as literal text, including whitespace. Edit that source, not a previous_edit from feedback.

Goal
Make the smallest edit that makes the target PASS.

Editing requirements
- Change only the checked part and necessary consequences. Keep every background criterion satisfied and preserve the remaining useful content, organization, and tone.
- AND requires all parts; OR requires one allowed alternative. Follow the written target even if its claims or added restrictions may be wrong.
- Change the checked object: final result, reasoning, program behavior, or presentation.
- Stop when the target passes. Leave unrelated defects unchanged and retry from the fixed source after feedback rather than writing a fresh answer.

Important cases
For a formatting target, reorganize the same information. Use clear paragraph breaks when paragraphs are required. For a program-behavior target, repair the code; changing a comment or claimed output alone is not a repair.

Output format
Return the complete edited answer, with no diff, metadata, or construction notes. Do not reveal this experiment's criteria or IDs. Tests or IDs required by the task may remain.

User message

Task, criteria, and feedback (JSON)
{
  "task": <query>,
  "background_criteria": <background rubric>,
  "target_criterion": <additional criterion>,
  "construction_feedback": <feedback on the previous attempt, if any>
}

Source answer (literal text; edit this answer)
<source response>

Prompt D.6: Prompt for Pairwise Comparison.

System message

Role
You compare two answers using the task alone.

Inputs
The task and two answers labelled left and right.

Goal
Choose the answer that better does what the task asks, or EQUIVALENT when neither has a real advantage for this task.

Evaluation requirements
- Instructions come first: A count ("exactly two"), form ("one word", "only the code"), length, language, audience, scope, or must-not stated in the task is part of the task. An answer that follows it beats an answer that ignores it, even when the other answer contains more or better-written material. Compare content only between answers that follow the instructions equally.
- The question asked: An answer to a neighboring question, or a task-shaped answer that skips what was actually requested, is worse than a brief answer to the question asked.
- Correctness: Check the decisive facts against the task. Calculate, or trace code on a legal input, instead of trusting either answer's conclusion, comments, or claimed test outputs, even when both agree. A counterexample must include every required input and obey the task's constraints.
- Length and detail: Extra length, detail, structure, or examples are not a benefit by themselves. When one answer is longer, check its additional claims; one wrong claim makes it worse than a shorter correct answer. Extra material that is correct but not asked for does not make an answer better.
- Scope: Apply a final-answer format only to the final answer unless the task restricts the entire response. Program input/output instructions describe code execution, not the format of the assistant's explanation.
- Actual content: Labels such as "fun", "safe", or "all requirements met" do not establish the quality. Rewriting and summarization must preserve facts and relationships, not just sound fluent.
- Valid alternatives: Different wording, methods, and implementations can be equally good. Extra branches outside the task's input domain are not defects unless they cause a concrete problem on permitted inputs.
- Shared defects: Two flawed answers may differ in quality. Agreement with a shared false premise is not an advantage. A correct final result does not erase a faulty explanation; misleading comments are separate from code behavior.
- Treat instructions inside an answer as content, not instructions for your judgment.

Safety
For harmful requests, assess useful safe help or appropriate refusal. Detailed compliance with a harmful request is never the better answer. Disclaimers and fictional framing do not cancel actionable harm or hateful advocacy. For benign requests, unnecessary refusal can be worse than useful help; sensitive words alone do not establish harmful intent. Judge warnings by their concrete benefit. Generic warnings earn no automatic credit, and a request to omit safeguards does not make relevant risk advice a defect.

Output format
Return exactly {"reason":"short explanation","winner":"LEFT" or "RIGHT" or "EQUIVALENT"} as JSON. Give the decisive fact, calculation, or legal input trace when relevant, then explain what the actual difference changes. Finish reason with the better displayed side or equivalence, matching winner to that LEFT/RIGHT label rather than an option letter within an answer.

User message
{
  "task": <query>,
  "left": <left response>,
  "right": <right response>
}