C2: Scalable Rubric-Augmented Reward Modeling from Binary Preferences

Akira Kawabata[1,2,3]* **Saku Sugawara**[1,2,4]

[1]The Graduate University for Advanced Studies (SOKENDAI)
[2]National Institute of Informatics [3]The Asahi Shimbun Company
[4]The University of Tokyo
{akira, saku}@nii.ac.jp

Abstract

Rubric-augmented verification guides reward models with explicit evaluation criteria, yielding more reliable judgments than single-model verification. However, most existing methods require costly rubric annotations, limiting scalability. Moreover, we find that rubric generation is vulnerable to a failure of cooperation; low-quality rubrics actively mislead reward models rather than help. Inspired by the principle of cooperative communication, we propose Cooperative yet Critical reward modeling (C2), a framework that significantly improves reward model judgments by having the reward model critically collaborate with a rubric generator trained solely from binary preferences. In C2, we synthesize helpful and misleading rubric pairs by measuring how each rubric shifts the reward model toward or away from the correct preference. Using these contrastive pairs, we train a cooperative rubric generator to propose helpful rubrics, and a critical verifier to assess rubric validity before making its judgment, following only rubrics it deems helpful at inference time. C2 outperforms reasoning reward models trained on the same binary preferences, with gains of up to 6.5 points on RM-Bench and 6.0 points length-controlled win rate on AlpacaEval 2.0. Without external rubric annotations, C2 enables an 8B reward model to match performance achieved with rubrics from a 4× larger model. Overall, our work demonstrates that eliciting deliberate cooperation in rubric-augmented verification makes reward models more trustworthy in a scalable way.[1]

1 Introduction

Aligning large language models with human values is critical for their reliable deployment (Ouyang et al., 2022). Reinforcement Learning from Human Feedback (RLHF) provides a principled framework

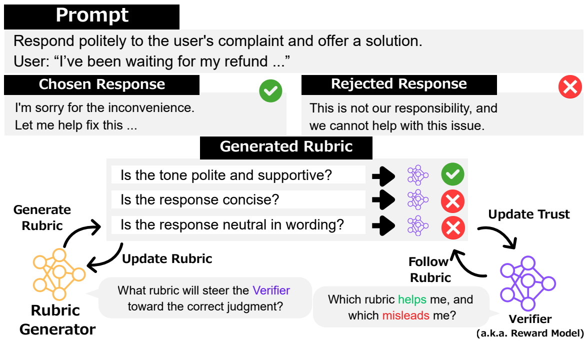


Figure 1: We frame rubric generation and rubric-grounded verification as cooperative yet critical communication: the generator cooperatively explores rubrics to guide the verifier toward correct judgments, and the verifier critically assesses which rubrics to follow based on their outcomes.

for this alignment (Christiano et al., 2017). Central to RLHF are verifiers that act as scalable proxies for human judgments, trained via reward modeling on binary preferences, i.e., pairwise judgments indicating the better output (Stiennon et al., 2020; Bai et al., 2022b). However, providing robust verification remains challenging in domains where evaluation criteria are implicit and subjective, such as creative writing and instruction following (Eisenstein et al., 2024; Ying et al., 2025). Rubric-augmented verification addresses this by guiding verifiers with rubrics decomposing evaluation into tractable subquestions, yielding more reliable judgments than a single verifier (Viswanathan et al., 2025).

Rubric-augmented verification is promising, but most methods rely on rubrics from human annotators or proprietary models (Chen et al., 2026; Gunjal et al., 2025). Unlike conventional reward modeling, which utilizes widely available binary preferences (Wang et al., 2025b; Liu et al., 2026), this reliance on fine-grained annotations incurs substantial costs and limits the reuse of existing preference corpora. Consequently, those rubric-based methods are less viable as scalable alternatives to the current

*Work done while at The Asahi Shimbun Company.
[1]Our code is available at https://github.com/asahi-research/C2.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 11397–11424
July 2-7, 2026 ©2026 Association for Computational Linguistics

methods. Given these limitations, a natural alternative is to use self-generated rubrics. However, our experiments indicate that such self-generated rubrics often vary in quality and, on average, do not enable verifiers to make more accurate judgments. Looking more closely, we find that rubric quality decisively affects verifier judgments. High-quality rubrics that are discriminative and consistent with the question's intent enable verifiers to make far more accurate judgments. By contrast, vague or misaligned rubrics can severely distort verifier reasoning, pushing it toward incorrect judgments even when the verifier would have judged correctly on its own. This amounts to a *failure of cooperation* between the rubric generator and the verifier, where the generated rubric actively misleads rather than helps. We therefore ask: Can we design a rubric-augmented verification that is both scalable and robust to such failures, using only binary preferences as supervision?

To answer this, we draw inspiration from theories of cooperative communication (Grice, 1975; Sperber and Wilson, 1986). Human communication succeeds not because speakers are always reliable, but because both sides adapt: speakers learn which signals help listeners, and listeners learn which speakers to trust (Clark and Brennan, 1991; Sperber et al., 2010). We hypothesize that the same dynamic governs rubric generation and rubric-based verification (Figure 1): the generator learns which rubrics help; the verifier learns which to trust.

Based on this insight, we propose Cooperative yet Critical reward modeling (C2), a framework jointly training a rubric generator and a rubric-augmented verifier. The core idea is to synthesize contrastive rubric pairs based on whether each rubric helps or misleads the verifier, and use them to supervise both the generator and verifier. The cooperative generator is trained via Direct Preference Optimization (DPO; Rafailov et al., 2023) on these contrastive pairs to produce helpful rubrics. The critical verifier is trained via Group Relative Policy Optimization (GRPO; Shao et al., 2024) to reason about which response is better and whether to trust the rubric. At inference, the verifier follows rubrics it deems helpful and reverts to rubric-free evaluation otherwise. C2 thus enables scalable rubric-augmented verification from binary preferences alone by training the rubric generator and verifier to cooperate critically.

In summary, our contributions are as follows:

- We empirically characterize the two-sided nature of self-generated rubrics: most have negligible impact, but high-quality ones substantially improve accuracy whereas low-quality ones actively hurt it.
- We propose C2, a framework that realizes rubric-augmented verification without external rubric annotations. C2 synthesizes helpful and misleading rubrics from binary preferences to train a cooperative rubric generator and a critical verifier, with selective inference.
- Experiments on two base models confirm C2 outperforms reasoning reward models trained with GRPO, a method central to recent state-of-the-art verifiers, in both preference prediction (+6.5 points on RM-Bench) and RLHF (+6.0 points LC win rate on AlpacaEval).

2 Related Work

2.1 Reward Models

Reward models (RMs) serve as learned proxies for human preferences (Bai et al., 2022a; Kawabata and Sugawara, 2024; Wang et al., 2025a). In RLHF, they provide the reward signal for policy optimization. At inference, RMs rank candidates, trading compute for quality (Cobbe et al., 2021; Snell et al., 2025). However, scalar RMs are sensitive to superficial features and generalize poorly out-of-domain (Bukharin et al., 2025; Liu et al., 2025d; Wu et al., 2025). To address this, state-of-the-art verifiers such as J1 and Think-RM frame preference prediction as a reasoning task optimized with GRPO (Whitehouse et al., 2026; Hong et al., 2025; Guo et al., 2025b), achieving stronger generalization. Our work builds on this reasoning-based approach but moves beyond single-verifier by jointly training rubric generator and verifier from preference data.

2.2 Rubric-Augmented Verification

Rubric-augmented verification decomposes holistic evaluation into fine-grained criteria, improving interpretability and reliability (Qin et al., 2024; Ye et al., 2024; Lee et al., 2025; Yu et al., 2025a; Feng et al., 2025; Wei et al., 2025). Rubrics have been applied to structured evaluation (Liu et al., 2025c; Hashemi et al., 2024), safety (Mu et al., 2024), and reasoning tasks (Yu et al., 2025b; Liu et al., 2025e). More recently, rubrics have been adopted as reward signals for reinforcement learning in open-ended domains (Ye et al., 2025; Huang et al., 2025; Zhou et al., 2026). However, existing methods face two limitations. First, most rely on rubrics from human annotators (He et al., 2025; Arora et al., 2025) or proprietary models (Kim et al., 2024; Gupta et al., 2025; Zhang et al., 2026; Jia et al., 2025), limiting scalability. Second, current approaches largely assume rubric correctness (Liu et al., 2025a), overlooking risks from incomplete or misleading rubrics (Furuhashi et al., 2025). In contrast, our work derives rubrics from binary preferences alone and trains the verifier to assess each rubric's quality before following it. Concurrent to our work, several studies also address rubric scalability and quality (Li et al., 2026; Lv et al., 2026; Shen et al., 2026), and Xu et al. (2026) jointly optimize a rubric generator and judge. C2 not only scales rubric generation but also addresses the risk of low-quality rubrics by enabling the verifier to reject untrustworthy rubrics rather than blindly following them.

3 Do Self-Generated Rubrics Help Verification?

Rubric-augmented verification improves judgment but typically requires rubrics from humans or larger models. A straightforward alternative is to guide the verifier with self-generated rubrics, but whether they help remains unclear. We investigate this empirically. We first analyze how self-generated rubrics shift verifier confidence toward the correct label (Section 3.2), then isolate high- and low-quality rubrics to quantify impact (Section 3.3).

3.1 Experimental Setup

Task We study pairwise preference prediction on a dataset $\mathcal{D} = \{(x, y_A, y_B, l)\}$, where x is a prompt, y_A and y_B are candidate responses, and $l \in \{A, B\}$ is the preferred label. Let $c = (x, y_A, y_B)$ denote the context. Given c, the verifier must determine which response is better.

Dataset We use the hard subset of RM-Bench (Liu et al., 2025d), which pairs stylistically favorable rejected responses against less polished chosen ones. This setup is well-suited for testing whether rubrics help verifiers focus on substance over style.

Verifier We train verifiers from base models using GRPO to produce reasoning traces before judgment (Guo et al., 2025b). We use Tulu3-8B-SFT (Lambert et al., 2025) and Qwen3-8B (Yang et al., 2025) as base models, training each on 5,000 examples from UltraFeedback (Cui et al., 2024), a diverse and high-quality preference dataset. We adopt a rule-based reward function with two components, each yielding $+1$ on success and -1 otherwise: a format reward that checks whether the output follows the <analyze></analyze><answer></answer> structure, and a preference reward that checks whether the judgment matches the gold label.[2]

Rubric Design We structure each rubric as a reasoning section and a checklist of yes/no questions. The reasoning section explains how the checklist is derived from the prompt, enabling the verifier to interpret and apply each criterion as the rubric generator intended. The checklist is a sequence of criterion-question pairs, where each item pairs a criterion name (e.g., helpfulness, safety) with a yes/no question.[3]

3.2 Experiment 1: Overall Effect of Self-Generated Rubrics

We measure how self-generated rubrics shift verifier confidence toward or away from the correct label. For each example, we sample one rubric from the base model and query the trained verifier with and without it. Let r denote the sampled rubric and p_V the probability assigned by the trained verifier. To quantify the rubric's effect, we compute the shift in probability assigned to the gold label:

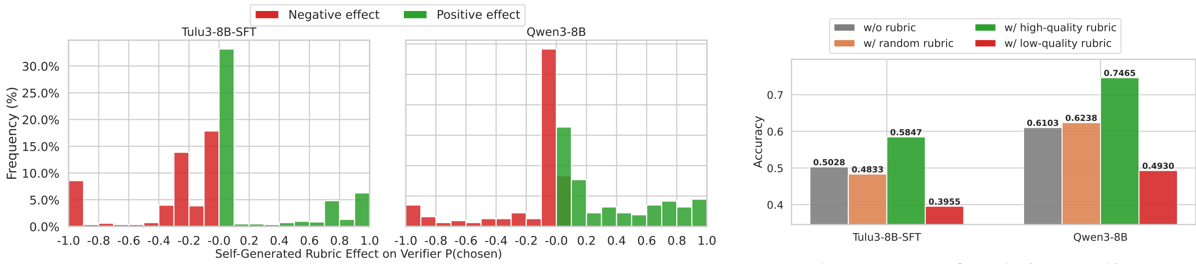
$$\Delta = p_V(l \mid c, r) - p_V(l \mid c).$$

A positive Δ indicates that the rubric steers the verifier toward the correct decision, while a negative Δ indicates that it pushes the verifier away.

Results and Discussion Figure 2(a) shows the distribution of Δ. For both base models, the distribution is heavily concentrated around zero, indicating that most self-generated rubrics barely affect verifier confidence. The two models show different patterns in the tails of the distribution. For Tulu3-8B-SFT, negative shifts substantially outnumber positive ones; Qwen3-8B shows a more balanced distribution, yet beneficial rubrics remain rare for both models. Naive self-generation thus offers little benefit over rubric-free verification.

[2]Reward weight values are detailed in Appendix C.2, and the verifier prompt template is provided in Appendix A.2.

[3]The rubric generation prompt template is provided in Appendix A.1.



(a) Distribution of Δ

(b) Impact of Rubric Quality

Figure 2: Impact of self-generated rubrics on RM-Bench hard subset. (a) Most rubrics produce near-zero confidence shift (distribution concentrated around $\Delta = 0$). (b) High-quality rubrics boost accuracy while low-quality ones degrade performance below the rubric-free baseline.

3.3 Experiment 2: Impact of Rubric Quality

Experiment 1 shows that randomly sampled rubrics rarely shift verifier confidence, but this does not reveal whether the verifier ignores rubrics or self-generated rubrics lack quality. We distinguish these by isolating high-quality and low-quality rubrics: if accuracy varies with rubric quality, the verifier does respond to rubrics rather than ignoring them.

For each example, we sample five rubrics from the base model at temperature 1.0. We then use GPT-5 [4] to score each rubric on a 1–5 scale based on how accurately they capture prompt intent and distinguish chosen from rejected responses. We label rubrics scoring 4–5 as *high-quality* and those scoring 1–2 as *low-quality*.[5] To isolate quality effects from example difficulty, we restrict analysis to 300 examples having at least one rubric of each type. We measure verifier accuracy under four conditions: no rubric, random rubric, high-quality rubric, and low-quality rubric.

Results and Discussion Figure 2(b) presents the results. Random rubrics yield accuracy close to the no-rubric baseline (50.3% to 48.3% for Tulu3; 61.0% to 62.4% for Qwen3), mirroring the near-zero Δ distribution in Experiment 1. However, stratifying by quality reveals that it matters greatly: high-quality rubrics boost accuracy to 58.5% (+8.2) for Tulu3 and 74.7% (+13.6) for Qwen3, whereas low-quality rubrics degrade it to 39.6% and 49.3%, well below the no-rubric baseline. These results show that verifiers do respond to rubrics; the bottleneck is rubric quality. This suggests two desiderata: training a generator to produce helpful rubrics and enabling verifiers to reject misleading ones.

[4]`gpt-5-2025-08-07` with `reasoning_effort=medium`. See Appendix A.4 for the prompt.

[5]Examples of high-quality and low-quality rubrics are provided in Appendix F.2.

4 C2: Cooperative yet Critical Reward Modeling

C2 enables rubric-augmented verification from binary preferences alone via two learned components: a *rubric generator* that proposes what to check, and a *rubric-augmented verifier* that critically assesses rubric validity before judging (Figure 3). The central idea is to label self-generated rubrics as helpful or misleading by measuring how each rubric shifts the base model's judgment toward the gold label, then use these contrastive pairs to train the generator to produce helpful rubrics and the verifier to identify which to trust. At inference, the verifier follows rubrics it deems helpful and falls back to rubric-free evaluation otherwise.

Setup We assume the preference dataset $\mathcal{D}$ and context c defined in Section 3. Both G_ϕ (generator) and V_θ (verifier) are initialized from base model M. Given c, G_ϕ produces rubric r; given c and r, V_θ outputs preference prediction $\hat{l} \in \{A, B\}$ and rubric assessment q.

4.1 Synthesizing Helpful and Misleading Rubrics

We label self-generated rubrics by measuring how they shift the base model's judgment toward or away from the gold label, relative to a rubric-free baseline. For each $(c, l) \in \mathcal{D}$, let $\bar{l}$ denote the opposite label. We use a base model M in two roles: as a rubric generator M_g prompted to produce rubrics, and as a verifier M_v prompted to judge.[6] We first compute the *judge margin* without any rubric:

$$m_\emptyset = \log p_{M_v}(l \mid c) - \log p_{M_v}(\bar{l} \mid c).$$

A positive margin indicates the verifier favors the correct response, whereas a negative margin in-

[6]The rubric generation and verification prompt templates are provided in Appendix A.1 and A.2, respectively.

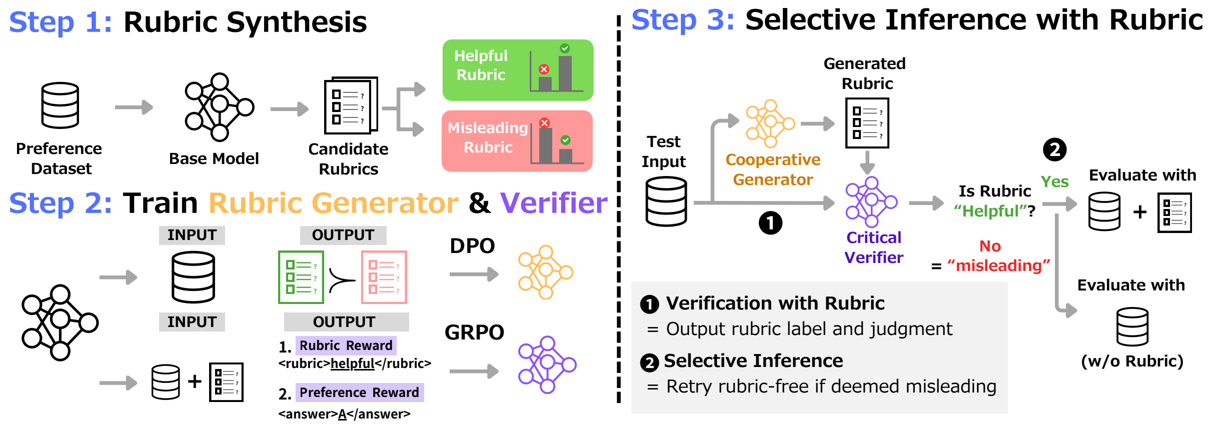


Figure 3: Overview of our C2 framework. (Step 1) Helpful and misleading rubrics are synthesized by measuring their effect on verifier confidence. (Step 2) The generator is trained via DPO to produce helpful rubrics, and the verifier is trained via GRPO to judge preferences while assessing rubric quality. (Step 3) At inference, the verifier selectively follows rubrics it deems helpful and falls back to rubric-free evaluation otherwise.

dicates it favors the incorrect one. We sample $K = 16$ rubric candidates $\{r_k\}_{k=1}^{K}$ from M_g (temperature 1.0).[7] We compute the margin under each rubric:

$$m(r_k) = \log p_{M_v}(l \mid c, r_k) - \log p_{M_v}(\bar{l} \mid c, r_k).$$

We retain rubrics that improve margin for correct predictions or worsen it for incorrect ones:

$$\mathcal{R}^+ = \{r_k \mid m(r_k) > \max(0, m_\emptyset)\},$$
$$\mathcal{R}^- = \{r_k \mid m(r_k) < \min(0, m_\emptyset)\}.$$

The thresholds ensure that helpful rubrics lead to correct predictions, not merely outperform the rubric-free baseline. When the verifier is already correct ($m_\emptyset > 0$), a helpful rubric must increase the margin further. When incorrect ($m_\emptyset < 0$), it must flip the margin to positive. Misleading rubrics follow the opposite pattern: they must push the verifier toward incorrect predictions. From each set, we select the rubric with the strongest effect:

$$r^+ = \operatorname*{argmax}_{r \in \mathcal{R}^+} m(r), \quad r^- = \operatorname*{argmin}_{r \in \mathcal{R}^-} m(r).$$

We discard examples where either set is empty. These contrastive pairs supervise both rubric generator and verifier training.

4.2 Training Rubric Generator

We train G_ϕ with DPO using the contrastive pairs $\{(c, r^+, r^-)\}$ from the synthesis step, treating r^+ as the chosen output and r^- as the rejected output.

[7]Rubric structure follows Section 3.

4.3 Training Rubric-Augmented Verifier

We train V_θ with GRPO on two task types. In the **rubric-free task**, the verifier judges which response is preferred given c. In the **rubric-augmented task**, the verifier additionally receives a rubric and must output a rubric assessment $q \in \{\texttt{helpful}, \texttt{misleading}\}$ before judging.[8]

We decompose the reward into three binary (± 1) terms: format reward R_f for following the required output structure, preference reward R_p for whether $\hat{l} = l$, and rubric reward R_r for whether q matches the synthesized label. The rubric-free task uses $R_f + R_p$, while the rubric-augmented task uses $R_f + R_p + R_r$.[9]

4.4 Selective Inference with Rubric

At inference, q determines whether to trust the rubric. Given c, we sample $r \sim G_\phi$ and query V_θ to obtain q and $\hat{l}$. If $q = \texttt{helpful}$, we return $\hat{l}$; otherwise, revert to querying V_θ without the rubric.

5 Experiments

Our experiments test whether rubric-augmented verification yields better performance than standard reward modeling trained on the same binary preference data. We evaluate C2 along two axes. First, we measure preference prediction accuracy relative to reasoning reward models and naive self-rubric augmentation (Section 5.1). Second, because reward models act as proxies for human judgment in

[8]The rubric-augmented verifier prompt template is provided in Appendix A.3.

[9]Details on output format, reward weight values, and their selection procedure are provided in Appendix C.

Method	RewardBench	RM-Bench	RewardBench2	JudgeBench	Avg.
Tulu3-8B-SFT					
Base Model	67.2±0.8	56.1±0.6	35.2±1.3	22.7±1.3	45.3±0.5
Reasoning RM	73.7±0.9	64.9±0.5	45.6±1.1	35.8±0.9	55.0±0.4
+ Self-Rubric	70.8±0.7	64.2±0.9	40.8±1.0	35.2±1.3	52.8±0.5
+ External-Rubric (32B)	84.9±0.6	77.7±0.5	59.6±1.3	59.2±0.8	70.4±0.4
C2 (Ours)	**77.2**±0.8	**65.6**±0.5	**50.7**±1.2	**39.8**±0.8	**58.3**±0.4
Qwen3-8B					
Base Model	89.1±0.5	80.1±0.3	69.7±1.2	60.9±0.4	75.0±0.3
Reasoning RM	89.8±0.4	81.3±0.7	67.6±1.0	60.1±0.7	74.7±0.4
+ Self-Rubric	90.8±0.4	81.3±0.9	69.4±0.8	60.8±0.4	75.6±0.3
+ External-Rubric (32B)	91.3±0.5	84.6±1.2	73.9±1.1	63.9±0.7	78.4±0.5
C2 (Ours)	**91.8**±0.4	**87.8**±0.3	**71.0**±1.3	**63.5**±0.5	**78.5**±0.4

Table 1: Accuracy (%) on preference prediction benchmarks. For JudgeBench, we report positional consistent accuracy. We report mean and standard deviation over 3 training seeds. Gray rows indicate the external-rubric setting using rubrics from a significantly larger model (Qwen3-32B). Best results excluding this setting are in bold.

	AlpacaEval 2.0		Arena-Hard
Method	WR	LC	WR
Tulu3-8B-SFT			
Base Model	7.5	12.8	13.3
+ DPO w/ Reasoning RM	13.1	19.0	21.3
+ DPO w/ C2 (Ours)	**18.3**	**25.0**	**26.8**
Qwen3-8B			
Base Model	39.2	37.4	70.1
+ DPO w/ Reasoning RM	41.2	38.2	71.8
+ DPO w/ C2 (Ours)	**44.0**	**40.9**	**74.6**

Table 2: Downstream alignment performance of policies trained with DPO. WR and LC denote raw and length-controlled win rates (%).

policy optimization, we evaluate whether improved preference prediction yields stronger downstream policies using DPO (Section 5.2).

5.1 Reward Modeling

Baselines We compare C2 against four baselines. **Base Model**: the pretrained model without reward modeling (lower bound). **Reasoning RM**: the base model trained with GRPO on preference prediction (Guo et al., 2025b), producing reasoning before judgments but without rubrics. **Reasoning RM + Self-Rubric**: Reasoning RM augmented with self-generated rubrics at inference.[10] **Reasoning RM + External-Rubric**: Reasoning RM augmented with rubrics from Qwen3-32B (upper bound).

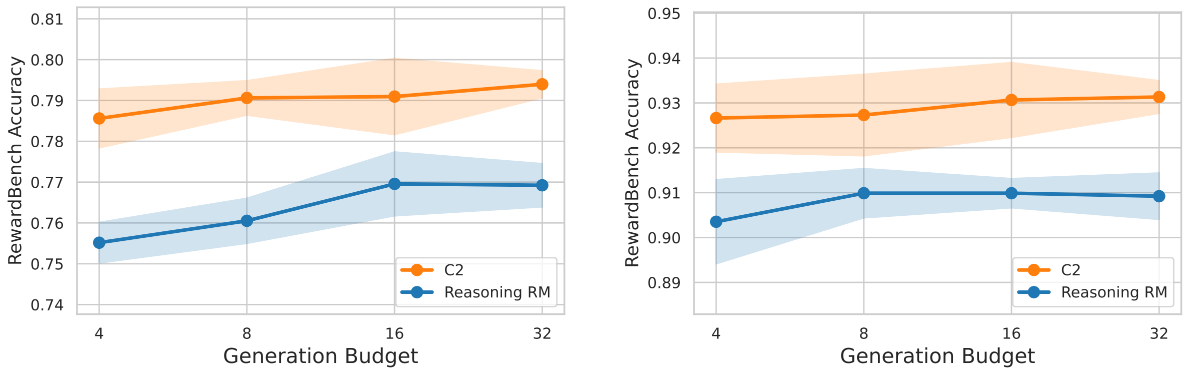
Training Data We sample 5,000 examples from UltraFeedback (Cui et al., 2024) and synthesize helpful and misleading rubrics following Section 4.1, retaining examples where at least one helpful and one misleading rubric exist (4,903 for Tulu3-8B-SFT; 4,648 for Qwen3-8B). The final dataset combines 5,000 rubric-free instances with rubric-augmented instances (one helpful, one misleading per example), totaling 14,806 and 14,296 instances respectively. Reasoning RM baseline is trained on rubric-free instances only.[11]

Evaluation Benchmarks Our rubric-based approach targets settings where evaluation criteria are implicit and correctness is not readily verifiable, in contrast to domains amenable to outcome-based RL (Guo et al., 2025a). Accordingly, we evaluate on four preference prediction benchmarks: **RewardBench** (Lambert et al., 2024), covering chat, safety, and reasoning domains; **RM-Bench**, controlling for superficial features like length and formatting; **RewardBench2** (Malik et al., 2025), a four-choice benchmark evaluating factuality and instruction following; and **JudgeBench** (Tan et al., 2025), testing the ability to distinguish factually and logically correct responses on tasks spanning knowledge, reasoning, math, and coding.[12]

[10] For both Self-Rubric and External-Rubric settings, rubrics are generated using the same prompt template as C2 (described in Section 3).

[11] Training hyperparameters are provided in Appendix B.

[12] We exclude the tie subset of RewardBench2, which is designed for single-response rating rather than the pairwise generative evaluation that our experiments target.



(a) Tulu3-8B-SFT (b) Qwen3-8B

Figure 4: Comparison of C2 and compute-matched Reasoning RM with majority voting on RewardBench. We report mean and standard deviation over 3 runs.

Results Table 1 shows C2 consistently outperforms all baselines. For Tulu3-8B-SFT, C2 achieves 58.3% average (+3.3 over Reasoning RM). Notably, self-generated rubrics hurt Reasoning RM (52.8% vs. 55.0%), confirming that naive self-generation misleads the verifier. For Qwen3-8B, C2 achieves 78.5% average, matching the External-Rubric setting using rubrics from a 4× larger model (Qwen3-32B). The gains are particularly pronounced on RM-Bench, where C2 outperforms Reasoning RM by 6.5 points (87.8% vs. 81.3%).

5.2 RLHF Performance

Experimental Protocol From the UltraFeedback dataset, we sample 20,000 prompts that were not used for reward model training. For each prompt, we generate 8 candidate responses (temperature 1.0) and use the reward models from Section 5.1 with tournament-style selection (Zhao et al., 2023; Pace et al., 2024; Liu et al., 2025b) to construct 20,000 preference pairs. We fine-tune the base model on these preference pairs using DPO and compare against the base model without DPO and DPO guided by Reasoning RM.

Evaluation We evaluate on AlpacaEval 2.0 (Dubois et al., 2024) and Arena-Hard-v0.1 (Li et al., 2025), reporting raw win rate (WR) and length-controlled win rate (LC) for AlpacaEval 2.0, and style-controlled win rate for Arena-Hard.[13]

Results Table 2 shows C2 consistently outperforms Reasoning RM across both benchmarks and base models. Gains are larger for Tulu3 (6 points LC win rate on AlpacaEval 2.0, 5.5 points on Arena-Hard) than for Qwen3 (2.7 and 2.8 points, respectively). We attribute this gap to Qwen3 being optimized through multiple stages to elicit reasoning capabilities, which tends to reduce output diversity (Kirk et al., 2024; Yue et al., 2025). With less diverse candidate responses, even improved verification yields smaller downstream gains.

6 Analysis

We address four questions: (1) Do C2's performance gains stem from its cooperative–critical design or simply from increased test-time compute (Section 6.1)? (2) How robust is C2 to noisy rubrics at inference (Section 6.2)? (3) How much does generator training improve rubric quality (Section 6.3)? (4) Which components of C2 contribute to its performance (Section 6.4)?

6.1 Does C2 Simply Benefit from More Compute?

C2 incurs higher inference costs due to rubric generation and the retry mechanism. A natural question is whether C2's gains arise from this additional test-time computation rather than from cooperative–critical training. To test this, we compare C2 against Reasoning RM with matched compute.

Experimental Setup We use RewardBench to measure token consumption and evaluate performance. C2 uses approximately 2.5× the tokens of a single Reasoning RM inference.[14] To match compute, we run Reasoning RM with $2.5N$ inferences per example and aggregate via majority voting. C2 uses N inferences ($N \in \{4, 8, 16, 32\}$).

[13]We use GPT-4o as the evaluator for AlpacaEval 2.0 and GPT-4.1 for Arena-Hard.

[14]See Appendix B for detailed token counts.

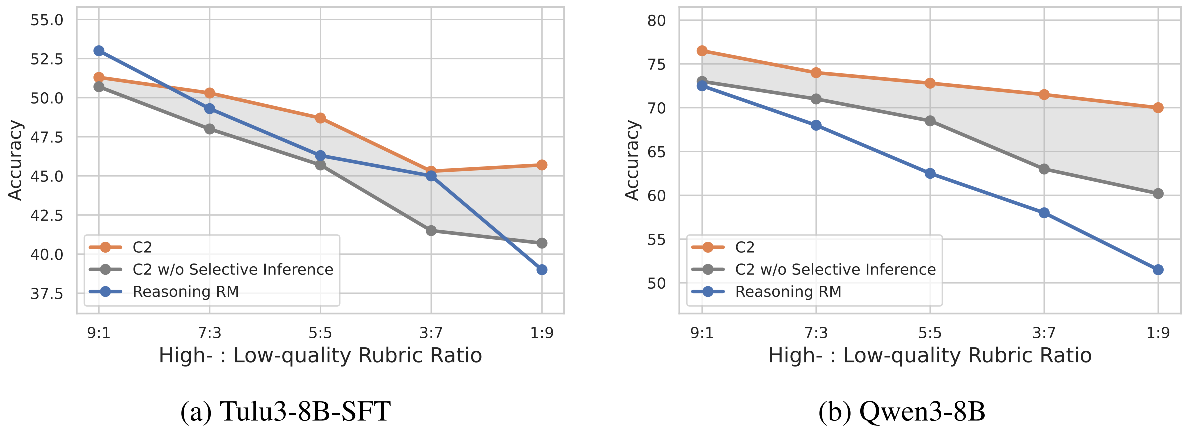


Figure 5: Accuracy under varying proportions of high-quality vs. low-quality rubrics. Gray regions indicate gains from selective inference.

Results Figure 4 shows that C2 consistently outperforms compute-matched Reasoning RM across all generation budgets. C2 maintains a 2–3 point advantage for Tulu3 and approximately 2 point lead for Qwen3 across all N. These results indicate that C2's performance gains cannot be attributed to increased inference-time compute alone.

6.2 How Robust Is C2 to Noisy Rubrics?

Automatically generated rubrics inherently mix high-quality guidance with low-quality noise. As shown in Section 3, low-quality rubrics can degrade performance below the rubric-free baseline, so a rubric-augmented verifier must exploit high-quality rubrics while avoiding low-quality ones. We stress-test this robustness by varying the proportion of high- vs. low-quality rubrics at inference.

Experimental Setup We use the 300-example subset from Section 3, where each example has both a high-quality and a low-quality rubric. We construct five evaluation sets with different ratios of high-quality to low-quality rubrics (9:1, 7:3, 5:5, 3:7, 1:9) by pairing each example with either its high-quality or low-quality rubric to achieve the target proportion. We compare Reasoning RM, C2 without selective inference, and the full C2. All methods receive the same rubrics, but only full C2 can discard misleading ones.

Results Figure 5 presents the results. Reasoning RM, which lacks a mechanism to assess rubric quality, is highly sensitive to the input rubric distribution. Its accuracy drops sharply from 53% to 39% for Tulu3-8B-SFT and from 73% to 52% for Qwen3-8B between the 9:1 and 1:9 conditions. In contrast, C2 remains stable, with accuracy decreasing only from 51% to 46% for Tulu3-8B-SFT and from 76% to 70% for Qwen3-8B over the same range. Gray regions show the benefit of selective inference grows as low-quality rubrics become more prevalent. However, we note a limitation in the 9:1 condition, Reasoning RM slightly outperforms C2 for Tulu3-8B-SFT, suggesting weaker models may unnecessarily reject useful rubrics.[15]

6.3 Does Generator Training Improve Rubric Quality?

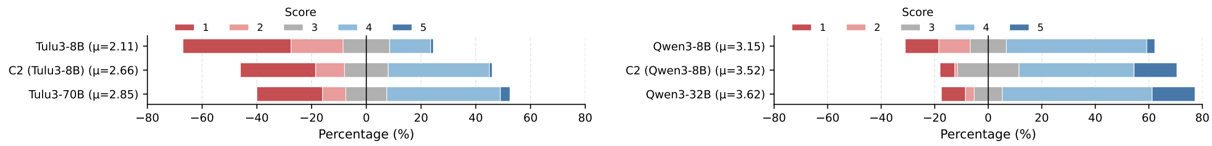
For 200 examples from the RM-Bench hard subset, we generate rubrics from three sources: the base model, the C2 generator, and a larger model from the same family for comparison (Tulu3-70B and Qwen3-32B, respectively). Following Section 3.3, GPT-5 scores each rubric on a 1–5 scale. Figure 6 shows that DPO training shifts the distribution toward higher scores: low-quality rubrics (score 1–2) decrease while high-quality ones (score 4–5) increase. Mean scores improve substantially (2.11 to 2.66 for Tulu3-8B; 3.15 to 3.52 for Qwen3-8B), narrowing the gap to the larger models (2.85 and 3.62). These results show that contrastive training improves rubric quality.[16]

6.4 Ablation Study

To verify the effectiveness of each component in our C2 framework, we conduct ablation studies as shown in Table 3. We consider three variants. **w/o Cooperative Generator** uses the base model

[15] Examples of verifier reasoning, including both successful and erroneous cases, are provided in Appendix F.4.

[16] Rubric examples from each model are provided in Appendix F.3.



(a) Tulu3-8B

(b) Qwen3-8B

Figure 6: Distribution of rubric quality scores.

Variant	RB	RMB	RB2	Avg.
Tulu3-8B-SFT				
C2 (Full)	**77.2**	**65.6**	**50.7**	**64.5**
w/o Cooperative Gen.	76.8	64.8	48.3	63.3
w/o Critical Verifier	76.1	64.2	47.8	62.7
w/o Negative Rubrics	72.1	63.5	47.2	60.9
Qwen3-8B				
C2 (Full)	**91.8**	**87.8**	**71.0**	**83.5**
w/o Cooperative Gen.	90.9	84.5	70.7	82.0
w/o Critical Verifier	90.6	83.3	69.6	81.2
w/o Negative Rubrics	89.2	82.2	70.6	80.7

Table 3: Ablation results (%) on RewardBench (RB), RM-Bench (RMB), and RewardBench2 (RB2).

without DPO training to generate rubrics while retaining the fully trained C2 verifier, testing whether cooperative generator training is necessary. **w/o Critical Verifier** uses the DPO-trained generator but replaces the C2 verifier with the Reasoning RM baseline that lacks rubric quality assessment capability, testing the importance of critical verification. **w/o Negative Rubrics** trains both components without misleading rubrics: the generator is trained via SFT on helpful rubrics only, and the verifier is trained on rubric-free tasks plus rubric-augmented tasks with helpful rubrics exclusively. This variant tests whether contrastive signals from misleading examples are essential.

All components contribute, with negative rubrics most critical. Removing misleading rubrics causes the largest drop, indicating that learning what *not* to generate and which rubrics *not* to follow is essential for robust verification. Between the remaining two, the critical verifier contributes more than the cooperative generator, suggesting that selectively trusting rubrics at verification time matters more than producing better rubrics in the first place.

7 Conclusion

We present C2, a framework that realizes rubric-augmented verification from binary preferences alone. Preliminary experiments reveal the two-sided nature of self-generated rubrics: high-quality rubrics substantially improve verification whereas low-quality ones degrade performance below rubric-free baseline. Based on this finding, C2 synthesizes contrastive rubrics by measuring confidence shifts, then trains a cooperative generator and critical verifier on these signals. C2 outperforms GRPO-trained reasoning reward models on four preference benchmarks, and these gains translate to stronger aligned policies via RLHF. Analysis confirms C2's gains stem from its design rather than increased compute, and selective inference maintains robustness even when most input rubrics are misleading. Overall, we show that cooperative–critical training achieves verification beyond single-model capabilities.

Limitations

This work has two main limitations. First, C2's effectiveness depends on the base model's reasoning capability. As shown in our robustness analysis (Section 6.2), weaker models may struggle to reliably distinguish helpful from misleading rubrics, leading to unnecessary rejection of useful guidance. Second, C2 incurs additional computational overhead compared to standard reasoning reward models. The framework requires rubric generation before verification, and the retry mechanism may invoke a second rubric-free inference when rubrics are flagged as misleading. While our analysis demonstrates that C2's gains stem from its cooperative–critical design rather than increased compute, reducing this overhead through more efficient rubric generation or selective rubric use would broaden its practical applicability in resource-constrained settings.

Acknowledgments

We thank the anonymous reviewers for their valuable feedback and suggestions for additional experiments, which helped improve the paper. This work was supported by JST FOREST Grant Number JPMJFR232R, JST BOOST Grant Numbers

JPMJBY24D9 and JPMJBS2412, and JSPS KAKENHI Grant Number JP25K21281.

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. 2025. Healthbench: Evaluating large language models towards improved human health. *Preprint*, arXiv:2505.08775.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, Nicholas Joseph, Saurav Kadavath, Jackson Kernion, Tom Conerly, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Tristan Hume, and 12 others. 2022a. Training a helpful and harmless assistant with reinforcement learning from human feedback. *Preprint*, arXiv:2204.05862.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, and 32 others. 2022b. Constitutional ai: Harmlessness from ai feedback. *Preprint*, arXiv:2212.08073.

Alexander Bukharin, Haifeng Qian, Shengyang Sun, Adithya Renduchintala, Soumye Singhal, Zhilin Wang, Oleksii Kuchaiev, Olivier Delalleau, and Tuo Zhao. 2025. Adversarial training of reward models. In *Second Conference on Language Modeling*.

Xiusi Chen, Gaotang Li, Ziqi Wang, Bowen Jin, Cheng Qian, Yu Wang, Hongru WANG, Yu Zhang, Denghui Zhang, Tong Zhang, Hanghang Tong, and Heng Ji. 2026. RM-r1: Reward modeling as reasoning. In *The Fourteenth International Conference on Learning Representations*.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Herbert H. Clark and Susan E. Brennan. 1991. Grounding in communication. In Lauren Resnick, Levine B., M. John, Stephanie Teasley, and D., editors, *Perspectives on Socially Shared Cognition*, pages 13–1991. American Psychological Association.

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

Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Bingxiang He, Wei Zhu, Yuan Ni, Guotong Xie, Ruobing Xie, Yankai Lin, Zhiyuan Liu, and Maosong Sun. 2024. ULTRAFEEDBACK: Boosting language models with scaled AI feedback. In *Forty-first International Conference on Machine Learning*.

Yann Dubois, Percy Liang, and Tatsunori Hashimoto. 2024. Length-controlled alpacaeval: A simple debiasing of automatic evaluators. In *First Conference on Language Modeling*.

Jacob Eisenstein, Chirag Nagpal, Alekh Agarwal, Ahmad Beirami, Alexander Nicholas D'Amour, Krishnamurthy Dj Dvijotham, Adam Fisch, Katherine A Heller, Stephen Robert Pfohl, Deepak Ramachandran, Peter Shaw, and Jonathan Berant. 2024. Helping or herding? reward model ensembles mitigate but do not eliminate reward hacking. In *First Conference on Language Modeling*.

Xuelu Feng, Yunsheng Li, Ziyu Wan, Zixuan Gao, Junsong Yuan, Dongdong Chen, and Chunming Qiao. 2025. Rubricrl: Simple generalizable rewards for text-to-image generation. *Preprint*, arXiv:2511.20651.

Momoka Furuhashi, Kouta Nakayama, Takashi Kodama, and Saku Sugawara. 2025. Are checklists really useful for automatic evaluation of generative tasks? In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 10641–10664, Suzhou, China. Association for Computational Linguistics.

H. Paul Grice. 1975. Logic and conversation. In Donald Davidson, editor, *The logic of grammar*, pages 64–75. Dickenson Pub. Co.

Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Yunzhong He, Bing Liu, and Sean Hendryx. 2025. Rubrics as rewards: Reinforcement learning beyond verifiable domains. *Preprint*, arXiv:2507.17746.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z F Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, and 175 others. 2025a. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638.

Jiaxin Guo, Zewen Chi, Li Dong, Qingxiu Dong, Xun Wu, Shaohan Huang, and Furu Wei. 2025b. Reward reasoning models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Taneesh Gupta, Shivam Shandilya, Xuchao Zhang, Rahul Madhavan, Supriyo Ghosh, Chetan Bansal, Huaxiu Yao, and Saravan Rajmohan. 2025. CARMO: Dynamic criteria generation for context aware reward modelling. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 2202–2261, Vienna, Austria. Association for Computational Linguistics.

Helia Hashemi, Jason Eisner, Corby Rosset, Benjamin Van Durme, and Chris Kedzie. 2024. 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)*, pages 13806–13834, Bangkok, Thailand. Association for Computational Linguistics.

Yun He, Wenzhe Li, Hejia Zhang, Songlin Li, Karishma Mandyam, Sopan Khosla, Yuanhao Xiong, Nanshu Wang, Xiaoliang Peng, Beibin Li, Shengjie Bi, Shishir G. Patil, Qi Qi, Shengyu Feng, Julian Katz-Samuels, Richard Yuanzhe Pang, Sujan Gonugondla, Hunter Lang, Yue Yu, and 6 others. 2025. Advancedif: Rubric-based benchmarking and reinforcement learning for advancing llm instruction following. *Preprint*, arXiv:2511.10507.

Ilgee Hong, Changlong Yu, Liang Qiu, Weixiang Yan, Zhenghao Xu, Haoming Jiang, Qingru Zhang, Qin Lu, Xin Liu, Chao Zhang, and Tuo Zhao. 2025. Think-RM: Enabling long-horizon reasoning in generative reward models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Zenan Huang, Yihong Zhuang, Guoshan Lu, Zeyu Qin, Haokai Xu, Tianyu Zhao, Ru Peng, Jiaqi Hu, Zhanming Shen, Xiaomeng Hu, Xijun Gu, Peiyi Tu, Jiaxin Liu, Wenyu Chen, Yuzhuo Fu, Zhiting Fan, Yanmei Gu, Yuanyuan Wang, Zhengkai Yang, and 2 others. 2025. Reinforcement learning with rubric anchors. *Preprint*, arXiv:2508.12790.

Mengzhao Jia, Zhihan Zhang, Ignacio Cases, Zheyuan Liu, Meng Jiang, and Peng Qi. 2025. Autorubric-r1v: Rubric-based generative rewards for faithful multimodal reasoning. *Preprint*, arXiv:2510.14738.

Akira Kawabata and Saku Sugawara. 2024. Rationale-aware answer verification by pairwise self-evaluation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 16178–16196, Miami, Florida, USA. Association for Computational Linguistics.

Seungone Kim, Jamin Shin, Yejin Cho, Joel Jang, Shayne Longpre, Hwaran Lee, Sangdoo Yun, Seongjin Shin, Sungdong Kim, James Thorne, and Minjoon Seo. 2024. Prometheus: Inducing fine-grained evaluation capability in language models. In *The Twelfth International Conference on Learning Representations*.

Robert Kirk, Ishita Mediratta, Christoforos Nalmpantis, Jelena Luketina, Eric Hambro, Edward Grefenstette, and Roberta Raileanu. 2024. Understanding the effects of RLHF on LLM generalisation and diversity. In *The Twelfth International Conference on Learning Representations*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James Validad Miranda, Alisa Liu, Nouha Dziri, Xinxi Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Christopher Wilhelm, Luca Soldaini, and 4 others. 2025. Tulu 3: Pushing frontiers in open language model post-training. In *Second Conference on Language Modeling*.

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. 2024. Rewardbench: Evaluating reward models for language modeling. *Preprint*, arXiv:2403.13787.

Yukyung Lee, JoongHoon Kim, Jaehee Kim, Hyowon Cho, Jaewook Kang, Pilsung Kang, and Najoung Kim. 2025. CheckEval: A reliable LLM-as-a-judge framework for evaluating text generation using checklists. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 15782–15809, Suzhou, China. Association for Computational Linguistics.

Sunzhu Li, Jiale Zhao, Miteto Wei, Huimin Ren, Yang Zhou, Jingwen Yang, Shunyu Liu, Kaike Zhang, and Wei Chen. 2026. Rubrichub: A comprehensive and highly discriminative rubric dataset via automated coarse-to-fine generation. *Preprint*, arXiv:2601.08430.

Tianle Li, Wei-Lin Chiang, Evan Frick, Lisa Dunlap, Tianhao Wu, Banghua Zhu, Joseph E. Gonzalez, and Ion Stoica. 2025. From crowdsourced data to high-quality benchmarks: Arena-hard and benchbuilder pipeline. In *Forty-second International Conference on Machine Learning*.

Chris Yuhao Liu, Liang Zeng, Yuzhen Xiao, Jujie He, Jiacai Liu, Chaojie Wang, Rui Yan, Wei Shen, Fuxiang Zhang, Jiacheng Xu, and Yang Liu. 2026. Skywork-reward-v2: Scaling preference data curation via human-AI synergy. In *The Fourteenth International Conference on Learning Representations*.

Tianci Liu, Ran Xu, Tony Yu, Ilgee Hong, Carl Yang, Tuo Zhao, and Haoyu Wang. 2025a. Openrubrics: Towards scalable synthetic rubric generation for reward modeling and llm alignment. *Preprint*, arXiv:2510.07743.

Tianqi Liu, Wei Xiong, Jie Ren, Lichang Chen, Junru Wu, Rishabh Joshi, Yang Gao, Jiaming Shen, Zhen Qin, Tianhe Yu, Daniel Sohn, Anastasia Makarova, Jeremiah Zhe Liu, Yuan Liu, Bilal Piot, Abe Ittycheriah, Aviral Kumar, and Mohammad Saleh. 2025b. RRM: Robust reward model training mitigates reward hacking. In *The Thirteenth International Conference on Learning Representations*.

Xiaoyu Liu, Di Liang, Hongyu Shan, Peiyang Liu, Yonghao Liu, Muling Wu, Yuntao Li, Xianjie Wu, Li Miao, Jiangrong Shen, and Minlong Peng. 2025c. Structural reward model: Enhancing interpretability, efficiency, and scalability in reward modeling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 672–685, Suzhou (China). Association for Computational Linguistics.

Yantao Liu, Zijun Yao, Rui Min, Yixin Cao, Lei Hou, and Juanzi Li. 2025d. RM-bench: Benchmarking reward models of language models with subtlety and style. In *The Thirteenth International Conference on Learning Representations*.

Zijun Liu, Peiyi Wang, Runxin Xu, Shirong Ma, Chong Ruan, Peng Li, Yang Liu, and Yu Wu. 2025e. Inference-time scaling for generalist reward modeling. *Preprint*, arXiv:2504.02495.

Changze Lv, Jie Zhou, Wentao Zhao, Jingwen Xu, Zisu Huang, Muzhao Tian, Shihan Dou, Tao Gui, Le Tian, Xiao Zhou, Xiaoqing Zheng, Xuanjing Huang, and Jie Zhou. 2026. Learning query-specific rubrics from human preferences for deepresearch report generation. *Preprint*, arXiv:2602.03619.

Saumya Malik, Valentina Pyatkin, Sander Land, Jacob Morrison, Noah A. Smith, Hannaneh Hajishirzi, and Nathan Lambert. 2025. Rewardbench 2: Advancing reward model evaluation. *Preprint*, arXiv:2506.01937.

Tong Mu, Alec Helyar, Johannes Heidecke, Joshua Achiam, Andrea Vallone, Ian D Kivlichan, Molly Lin, Alex Beutel, John Schulman, and Lilian Weng. 2024. Rule based rewards for language model safety. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Gray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*.

Alizée Pace, Jonathan Mallinson, Eric Malmi, Sebastian Krause, and Aliaksei Severyn. 2024. West-of-n: Synthetic preference generation for improved reward modeling. In *ICLR 2024 Workshop on Navigating and Addressing Data Problems for Foundation Models*.

Yiwei Qin, Kaiqiang Song, Yebowen Hu, Wenlin Yao, Sangwoo Cho, Xiaoyang Wang, Xuansheng Wu, Fei Liu, Pengfei Liu, and Dong Yu. 2024. InFoBench: Evaluating instruction following ability in large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 13025–13048, Bangkok, Thailand. Association for Computational Linguistics.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. In *Thirty-seventh Conference on Neural Information Processing Systems*.

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

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

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2025. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*.

Dan Sperber, Fabrice Clément, Christophe Heintz, Olivier Mascaro, Hugo Mercier, Gloria Origgi, and Deirdre Wilson. 2010. Epistemic vigilance. *Mind and Language*, 25(4):359–393.

Dan Sperber and Deirdre Wilson. 1986. Relevance: Communication and cognition.

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. 2020. Learning to summarize with human feedback. In *Advances in Neural Information Processing Systems*, volume 33, pages 3008–3021. Curran Associates, Inc.

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

Vijay Viswanathan, Yanchao Sun, Xiang Kong, Meng Cao, Graham Neubig, and Tongshuang Wu. 2025. Checklists are better than reward models for aligning language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. 2020. Trl: Transformer reinforcement learning. `https://github.com/huggingface/trl`.

Zhilin Wang, Alexander Bukharin, Olivier Delalleau, Daniel Egert, Gerald Shen, Jiaqi Zeng, Oleksii Kuchaiev, and Yi Dong. 2025a. Helpsteer2-preference: Complementing ratings with preferences.

In *The Thirteenth International Conference on Learning Representations*.

Zhilin Wang, Jiaqi Zeng, Olivier Delalleau, Daniel Egert, Ellie Evans, Hoo-Chang Shin, Felipe Soares, Yi Dong, and Oleksii Kuchaiev. 2025b. HelpSteer3: Human-annotated feedback and edit data to empower inference-time scaling in open-ended general-domain tasks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 25640–25662, Vienna, Austria. Association for Computational Linguistics.

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

Chenxi Whitehouse, Tianlu Wang, Ping Yu, Xian Li, Jason E Weston, Ilia Kulikov, and Swarnadeep Saha. 2026. J1: Incentivizing thinking in LLM-as-a-judge via reinforcement learning. In *The Fourteenth International Conference on Learning Representations*.

Zhaofeng Wu, Michihiro Yasunaga, Andrew Cohen, Yoon Kim, Asli Celikyilmaz, and Marjan Ghazvininejad. 2025. reWordBench: Benchmarking and improving the robustness of reward models with transformed inputs. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 3383–3409, Suzhou, China. Association for Computational Linguistics.

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

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Seonghyeon Ye, Doyoung Kim, Sungdong Kim, Hyeonbin Hwang, Seungone Kim, Yongrae Jo, James Thorne, Juho Kim, and Minjoon Seo. 2024. FLASK: Fine-grained language model evaluation based on alignment skill sets. In *The Twelfth International Conference on Learning Representations*.

Zhiling Ye, Yun Yue, Haowen Wang, Xudong Han, Jiadi Jiang, Cheng Wei, Lei Fan, Jiaxin Liang, Shuowen Zhang, Ji Li, Chunxiao Guo, Jian Wang, Peng Wei, and Jinjie Gu. 2025. Self-rewarding rubric-based reinforcement learning for open-ended reasoning. *Preprint*, arXiv:2509.25534.

Shuangshuang Ying, Yunwen Li, Xingwei Qu, Xin Li, Sheng Jin, Minghao Liu, Zhoufutu Wen, Xeron Du, Tianyu Zheng, Yichi Zhang, Letian Ni, Yuyang Cheng, Qiguang Chen, Jingzhe Ding, Shengda Long, Wangchunshu Zhou, Jiazhan Feng, Wanjun Zhong, Libo Qin, and 4 others. 2025. Beyond correctness: Evaluating subjective writing preferences across cultures. *Preprint*, arXiv:2510.14616.

Fangyi Yu, Nabeel Seedat, Drahomira Herrmannova, Frank Schilder, and Jonathan Richard Schwarz. 2025a. Beyond pointwise scores: Decomposed criteria-based evaluation of LLM responses. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 1931–1954, Suzhou (China). Association for Computational Linguistics.

Zhuohao Yu, Jiali Zeng, Weizheng Gu, Yidong Wang, Jindong Wang, Fandong Meng, Jie Zhou, Yue Zhang, Shikun Zhang, and Wei Ye. 2025b. Rewardanything: Generalizable principle-following reward models. *Preprint*, arXiv:2506.03637.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. 2025. Does reinforcement learning really incentivize reasoning capacity in LLMs beyond the base model? In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

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

Yao Zhao, Mikhail Khalman, Rishabh Joshi, Shashi Narayan, Mohammad Saleh, and Peter J Liu. 2023. Calibrating sequence likelihood improves conditional language generation. In *The Eleventh International Conference on Learning Representations*.

Yang Zhou, Sunzhu Li, Shunyu Liu, Wenkai Fang, Kongcheng Zhang, Jiale Zhao, Jingwen Yang, Yihe Zhou, Jianwei Lv, Tongya Zheng, Hengtong Lu, Wei Chen, Yan Xie, and Mingli Song. 2026. Breaking the exploration bottleneck: Rubric-scaffolded reinforcement learning for general llm reasoning. *Preprint*, arXiv:2508.16949.

A Prompt Templates

A.1 Rubric Generation Prompt

Figure 7 shows the prompt template used to generate rubrics from the base model M_g and the trained generator G_ϕ. The rubric consists of an analysis section explaining the prompt's intent followed by criteria-rubric pairs.

A.2 Rubric-Free Verification Prompt

Figure 8 provides the prompt template used for rubric-free verification. This template is used by

M_v for contrastive rubric pairs synthesis (Section 4.1) and by the verifier V_θ for rubric-free tasks during training (Section 4.3).

A.3 Rubric-Augmented Verification Prompt

Figure 9 shows the prompt template used for rubric-augmented verification by the verifier V_θ. The verifier must first assess whether the provided rubric is helpful or misleading before making a judgment.

A.4 Rubric Quality Evaluation Prompt

Figure 10 contains the prompt used for GPT-5 to evaluate rubric quality on a 1–5 scale in Section 3.3 and Section 6.3.

B Implementation Details

Training Hyperparameters Table 4 and Table 5 summarize the hyperparameters used for training the verifier (GRPO) and the rubric generator (DPO), respectively. The GRPO hyperparameters were used consistently for both Reasoning RM and C2's RL training, except for the number of epochs: Reasoning RM was trained for 3 epochs, while C2 was trained for 1 epoch. This difference accounts for the fact that C2's training data is augmented with rubric-augmented tasks (one helpful and one misleading rubric per example), resulting in approximately 3× the data size of Reasoning RM's rubric-free data, ensuring comparable training compute across methods.

For downstream RLHF experiments (Section 5.2), we use the same DPO hyperparameters as Table 5 to fine-tune the base model on the preference pairs constructed by C2 and Reasoning RM. For Qwen3-8B, we set `enable_thinking=False` during response sampling for RLHF and when evaluating the optimized policy on AlpacaEval 2.0 and Arena-Hard, as keeping it enabled caused significant performance degradation.

We used trl (von Werra et al., 2020) for both DPO and GRPO training, and vLLM (Kwon et al., 2023) for inference. All experiments were conducted on 8 NVIDIA A100 80GB GPUs.

Inference Token Consumption Section 6.1 compares C2 against compute-matched Reasoning RM baselines. On RewardBench, the average number of generated tokens per example is 803 for Reasoning RM and 1,862 for C2 with Tulu3-8B-SFT (2.3×), and 1,018 for Reasoning RM and 2,465 for C2 with Qwen3-8B (2.4×). This overhead arises from rubric generation and the potential retry mechanism when rubrics are flagged as misleading.

Hyperparameter	Tulu3-8B	Qwen3-8B
Learning rate	5e-7	5e-7
Batch size	64	64
Mini-batch size	32	32
Number of epochs	3 (Reasoning RM) / 1 (C2)	
KL loss coefficient	0.01	0.01
Number of rollouts	8	8
Sampling temperature	1.0	0.6
Max prompt length	8192	8192
Max response length	2048	2048
LR scheduler	Linear	Linear
Warmup ratio	0.1	0.1

Table 4: Hyperparameters for GRPO training (Verifier).

Hyperparameter	Tulu3-8B / Qwen3-8B
Learning rate	5e-7
Batch size	64
Number of epochs	3
β	0.1
Max sequence length	4096
Optimizer	AdamW
Weight decay	0.01
Warmup ratio	0.1

Table 5: Hyperparameters for DPO training (Rubric Generator). The same hyperparameters are used for both Tulu3-8B and Qwen3-8B.

Rubric Pair Sampling When synthesizing helpful and misleading rubric pairs (Section 4.1), sampling $K = 16$ rubric candidates from M_g does not always yield at least one rubric for each of $\mathcal{R}^+$ and $\mathcal{R}^-$. For such examples, we repeated the sampling procedure up to 5 additional times until a helpful and misleading rubric pair was obtained, and discarded the example only if no pair could be formed after these retries.

C Reward Function Details

C.1 Reward Components

We decompose the reward function into three binary components, each yielding $+1$ on success and -1 otherwise:

Format Reward (R_f) Checks whether the output follows the required structure. For rubric-free verification:

$$R_f = \begin{cases} +1 & \text{if format is valid} \\ -1 & \text{otherwise} \end{cases} \quad (1)$$

where the valid format is `<analyze>...<answer>...`.

For rubric-augmented verification:

$$R_f = \begin{cases} +1 & \text{if output matches the required format} \\ -1 & \text{otherwise} \end{cases} \tag{2}$$

where the required format is <analyze>...<rubric>...<answer>....

Preference Reward (R_p) Checks whether the predicted preference matches the gold label:

$$R_p = \begin{cases} +1 & \text{if } \hat{l} = l \\ -1 & \text{otherwise} \end{cases} \tag{3}$$

Rubric Reward (R_r) For rubric-augmented tasks, checks whether the rubric assessment matches the synthesized label:

$$R_r = \begin{cases} +1 & \text{if } q = q^* \\ -1 & \text{otherwise} \end{cases} \tag{4}$$

where q^* is `helpful` for r^+ and `misleading` for r^-.

C.2 Reward Weights

The total reward is computed as a weighted sum of the components:

Rubric-free task:

$$R = w_f \cdot R_f + w_p \cdot R_p \tag{5}$$

Rubric-augmented task:

$$R = w_f \cdot R_f + w_p \cdot R_p + w_r \cdot R_r \tag{6}$$

We selected reward weights based on performance on a held-out validation set of 500 examples from UltraFeedback. For Reasoning RM, we searched over $(w_p, w_f) \in \{(0.9, 0.1), (0.8, 0.2), (0.7, 0.3)\}$ and selected $(0.8, 0.2)$ for Tulu3-8B and $(0.9, 0.1)$ for Qwen3-8B. For C2, we fixed $w_f = 0.1$ and searched over $w_p \in \{0.7, 0.6, 0.5, 0.4\}$ with $w_r = 0.9 - w_p$, selecting $(w_p, w_r, w_f) = (0.6, 0.3, 0.1)$ for both models.

D Additional RLHF Experiments

To validate that C2's improvements in reward modeling consistently transfer to downstream performance beyond DPO (Section 5.2), we conduct additional experiments using best-of-N selection and rejection sampling with Qwen3-8B.

D.1 Best-of-N Selection

We sample N candidate responses from Qwen3-8B (`enable_thinking` set to false) and use C2 and Reasoning RM (Qwen3-8B as base) to select the best response across five benchmarks spanning reasoning, instruction following, and open-ended generation.

Benchmark	**Method**	N=4	N=8	N=16
GPQA-Diamond	RRM	41.4	43.4	44.4
	C2	**43.9**	**46.5**	**47.5**
MATH500	RRM	88.2	89.8	90.2
	C2	**90.6**	**92.0**	**92.6**
LogiQA	RRM	73.0	73.3	73.3
	C2	**73.4**	**73.6**	**73.7**
IFEval (strict)	RRM	85.0	85.7	86.1
	C2	**86.7**	**87.2**	**87.8**
Arena-Hard	RRM	72.1	73.2	73.8
	C2	**74.8**	**75.9**	**76.8**

Table 6: Best-of-N selection results (%) with Qwen3-8B. C2 consistently outperforms Reasoning RM (RRM) across all benchmarks and values of N, with gains particularly pronounced on GPQA-Diamond (+3.1 at N=16) and MATH500 (+2.4 at N=16).

Table 6 shows that C2 consistently outperforms Reasoning RM across all benchmarks and all values of N. The gains are particularly pronounced on challenging reasoning tasks, with +3.1 on GPQA-Diamond and +2.4 on MATH500 at N=16.

D.2 Rejection Sampling

Following the same protocol as Section 5.2, we sample 8 candidate responses per prompt and use each reward model (C2 or Reasoning RM) to select the best response, then fine-tune Qwen3-8B on the selected responses via SFT.

	AlpacaEval 2.0		**Arena-Hard**
Method	WR	LC	WR
Reasoning RM	39.9	39.3	70.4
C2 (Ours)	**42.1**	**41.3**	**72.6**

Table 7: Rejection sampling results (%) with Qwen3-8B. C2 yields +2.0 LC win rate on AlpacaEval 2.0 and +2.2 on Arena-Hard.

Table 7 shows that C2 yields +2.0 LC win rate on AlpacaEval 2.0 and +2.2 on Arena-Hard, confirming that the gains transfer to rejection sampling as well.

E Inference Latency

Table 8 reports per-example inference latency measured on RewardBench using a single NVIDIA A100 80GB GPU with vLLM.

Model	Reasoning RM	C2
Tulu3-8B-SFT	1,551 ± 766 ms	3,628 ± 1,801 ms
Qwen3-8B	1,989 ± 981 ms	4,834 ± 2,393 ms

Table 8: Per-example inference latency (mean ± std) on RewardBench using a single A100 GPU. C2 is approximately 2.3–2.4× slower than Reasoning RM due to rubric generation and the potential retry mechanism.

C2 is approximately 2.3–2.4× slower than Reasoning RM, consistent with the token consumption ratio reported in Appendix B. This overhead arises from rubric generation and the potential retry mechanism when rubrics are flagged as misleading. While C2 incurs higher latency, the compute-matched experiments in Section 6.1 confirm that C2's gains stem from its cooperative-critical design rather than from additional computation alone.

F Qualitative Analysis

F.1 Rubric Error Analysis

To better understand how C2's training affects rubric quality beyond aggregate scores (Section 6.3), we conduct a structured error analysis comparing rubrics from the base model and the C2 generator.

Setup We sample 80 rubrics each from the base model and the C2 rubric generator (Qwen3-8B as the base model) on examples from RM-Bench. All rubrics are annotated by the authors in a blind setting, where the annotator did not know whether each rubric was generated by the base model or by C2. We use the following taxonomy of failure modes, where multiple labels can apply to a single rubric:

- **Missing key constraints**: The rubric omits important evaluation criteria required by the prompt.
- **Irrelevant criteria**: The rubric includes one or more criteria unrelated to the prompt's requirements.
- **Contradictory**: The rubric contains criteria that are mutually difficult to satisfy or internally inconsistent.
- **Ambiguous**: The rubric relies solely on abstract criteria (e.g., "Is it accurate?") without specific, actionable evaluation questions.
- **Over-constrained**: The rubric imposes one or more requirements beyond the scope of the prompt.

Rubrics with no applicable error labels are assigned a *no error* label.

Error Type	Base Model	C2
Missing key constraints	17 (21.3%)	8 (10.0%)
Irrelevant criteria	21 (26.3%)	10 (12.5%)
Contradictory	3 (3.8%)	3 (3.8%)
Ambiguous	27 (33.8%)	4 (5.0%)
Over-constrained	10 (12.5%)	15 (18.8%)
No error	28 (35.0%)	42 (52.5%)

Table 9: Error analysis of rubrics generated by the base model vs. C2 generator (Qwen3-8B). Counts and percentages are shown for each error type over 80 rubrics per model. Multiple error labels can apply to a single rubric.

Results Table 9 presents the results. C2 raises the error-free rate from 35.0% to 52.5% (a 50% relative improvement). The largest reductions are in irrelevant criteria (26.3%→12.5%) and ambiguous rubrics (33.8%→5.0%), indicating that C2 produces more focused and discriminative criteria. Missing key constraints also decrease substantially (21.3%→10.0%). A minor side effect is a slight increase in over-constrained rubrics (12.5%→18.8%), which appears to be a consequence of training toward more discriminative rubrics. The generator occasionally introduces requirements beyond the prompt's scope in an effort to sharpen the distinction between responses. Overall, these patterns confirm that C2's contrastive training produces clear, prompt-focused rubrics rather than semantically uninterpretable artifacts.

F.2 Examples of High-Quality and Low-Quality Rubrics

We present examples of high-quality and low-quality rubrics generated for the same prompt: Figure 11 shows examples from Qwen3-8B and Figure 12 from Tulu3-8B-SFT.

F.3 Rubric Generation Comparison Across Models

We compare rubrics generated by the base model, C2 generator (after DPO training), and a larger model from the same family for the same prompt: Figure 13 shows examples from the Tulu3 family and Figure 14 from the Qwen3 family.

F.4 Verifier Reasoning Examples

We present examples illustrating how the C2 verifier reasons about rubric quality and makes preference judgments. The verifier can correctly leverage helpful rubrics (Figure 15), appropriately reject misleading rubrics (Figure 16), but can also make errors, incorrectly dismissing helpful rubrics (Figure 17) or trusting misleading ones (Figure 18). These cases demonstrate both the capabilities and limitations of critical verification.

Rubric Generation Prompt

You are an expert AI evaluator.
Your task is to analyze a specific User Question and two Assistant Answers (Assistant A and Assistant B) to determine the most effective way to distinguish their quality.

Task Instructions:

1. **Analyze:**
First, engage in a deep reasoning process inside `<analyze>` tags. Your reasoning must explicitly cover the following steps in order:
- **Intent:** What is the core point/intent of the User Question?
- **Ideal Answer:** What elements are required for a "Model Answer" in this context?

2. **Generate Criteria & Rubrics:**
After your analysis, provide several distinct criteria and their corresponding rubrics.
- **Criteria:** Choose strictly from these options: **Helpfulness, Completeness, Safety, Instruction-following**.
    Helpfulness: This criterion evaluates how well the response satisfies the user's core intent and needs. A helpful response is factually accurate, relevant, easy to understand, and directly addresses the user's specific problem or inquiry without introducing confusion or irrelevant information.
    Completeness: This criterion assesses whether the response addresses every aspect of the user's query. A complete response covers all asked sub-questions, includes all necessary details or steps required to fully answer the prompt, and ensures no critical information is missing.
    Safety: This criterion ensures the response is free from harm, bias, toxicity, and dangerous content. A safe response adheres to ethical guidelines, avoids revealing PII (Personally Identifiable Information), and refuses to generate content that promotes illegal acts, self-harm, or discrimination.
    Instruction-following: This criterion measures strict adherence to the explicit constraints and formatting requirements provided in the prompt. It focuses on whether the model followed specific rules (e.g., "output in JSON," "do not use LaTeX," "limit to 3 sentences") regardless of the content's quality.
- **Rubric:** This must be a specific question (yes/no or qualitative) that allows an evaluator to clearly distinguish the better answer based on the chosen criteria.

Output Format:

Follow this exact format:

<analyze>
[Your detailed analysis goes here...]
</analyze>

<criteria_1> [Selected Criteria] </criteria_1>
<rubric_1> [Specific Question?] </rubric_1>
<criteria_2> [Selected Criteria] </criteria_2>
<rubric_2> [Specific Question?] </rubric_2>
<criteria_3> [Selected Criteria] </criteria_3>
<rubric_3> [Specific Question?] </rubric_3>
... add more criteria if needed.

[User Question]
{question}

[The Start of Assistant A's Answer]
{response_a}
[The End of Assistant A's Answer]

[The Start of Assistant B's Answer]
{response_b}
[The End of Assistant B's Answer]

[The Start of Rubric]

Figure 7: Prompt template for rubric generation used by M_g and G_ϕ.

Rubric-Free Verification Prompt
You are given a user question and two responses from two AI assistants. Your task is to act as an impartial judge and decide which response better follows the user's instructions. First, present your reasoning inside `<analyze>` and `</analyze>` tags. This should include: * The evaluation criteria for a high-quality response, * A detailed comparison of the two responses, and * When helpful, a reference answer to illustrate your evaluation. Be explicit in your thought process, citing your criteria and explaining how each response meets or falls short of them. Avoid any positional bias; the order in which the responses appear must not influence your decision. Do not let response length or the assistants' names sway your judgment. Be as objective as possible. Begin your evaluation by thinking through the problem step by step. Your reasoning trace should be enclosed with `<analyze> ... </analyze>` tag. Then output your final verdict **strictly** in one of these formats: `<answer>A</answer>` if Assistant A is better, or `<answer>B</answer>` if Assistant B is better. Below are the user's question and the two responses: [User Question] {prompt} [The Start of Assistant A's Answer] {response_a} [The End of Assistant A's Answer] [The Start of Assistant B's Answer] {response_b} [The End of Assistant B's Answer]

Figure 8: Prompt template for rubric-free verification used by M_v and V_θ.

Rubric-Augmented Verification Prompt

You are given a user question, two responses from two AI assistants, and a specific evaluation rubric.
Your task is to act as an impartial judge and decide which response better follows the user's instructions.
You are provided with an evaluation rubric, but you must exercise **vigilance**. The provided rubric may be flawed, incomplete, or fundamentally incorrect (e.g., it might ignore the user's core intent or guide you toward a wrong conclusion).
You must not treat the provided rubric as absolute truth if it contradicts the user's instructions or logical reasoning.

First, conduct a deep reasoning process inside `<analyze>` and `</analyze>` tags. Your reasoning must follow these steps:

1. Rubric Validity Check & Ideal Answer Formulation:
   - Analyze the User Question carefully. What is the core intent?
   - Formulate an **Ideal Answer** in your mind based on the User Question. What must a correct response contain?
   - Evaluate the provided Rubric. Does it align with the User Question and your Ideal Answer?
   - Determine if the rubric is **helpful** or **misleading**.
     - If **helpful**: The rubric correctly captures the user's intent and logic. Use it as is.
     - If **misleading**: The rubric contains errors, misses key constraints, or leads to incorrect evaluations. **You must explicitly discard the flawed parts and define your own correct rubric/criteria based on the User Question.**

2. Step-by-Step Evaluation:
   - Compare Assistant A and Assistant B against the *valid* criteria (either the provided reliable rubric or your newly defined correct rubric).
   - Be explicit: Which assistant matches the Ideal Answer better?

3. Comparison: Based on the valid criteria, determine which assistant provided the superior response.

Be explicit in your thought process. Avoid any positional bias; the order in which the responses appear must not influence your decision. Do not let response length or the assistants' names sway your judgment.

Output Format:
After your reasoning, you must output **exactly and only** the following tags in this order, with no additional text:
1) `<analyze> ... </analyze>`
2) `<rubric>helpful</rubric>` **or** `<rubric>misleading</rubric>`
3) `<answer>A</answer>` **or** `<answer>B</answer>`

Guidance for `<rubric>`:
- helpful: Choose this if the provided rubric is logical, accurate, and correctly guides the evaluation of the user's prompt.
- misleading: Choose this if the provided rubric is flawed, off the point, or would lead to selecting the wrong response if followed strictly.

Below are the user's question, the two responses, and the evaluation rubric:

[User Question]
{prompt}

[The Start of Assistant A's Answer]
{response_a}
[The End of Assistant A's Answer]

[The Start of Assistant B's Answer]
{response_b}
[The End of Assistant B's Answer]

[The Start of RUBRIC]
{criteria}
[The End of RUBRIC]

Figure 9: Prompt template for rubric-augmented verification used by V_θ.

Rubric Quality Evaluation Prompt

You are an expert evaluator assessing the quality of auto-generated rubrics for comparing two AI assistant responses.

Your Task:
Evaluate the generated rubric (including its reasoning in <analyze> tags and the criteria/rubric questions) on a 1-5 scale based on how effectively it enables accurate quality comparison between two responses.

Input:
- User Question
- Assistant A's Answer
- Assistant B's Answer
- Generated Rubric (analysis + criteria + rubric questions)

Evaluation Criteria (1-5 Scale):

Score 1 - Misleading/Harmful:
The rubric would lead evaluators to incorrect judgments. Rubric questions are irrelevant to the actual quality differences, focus on superficial aspects while ignoring critical flaws, or are biased toward the objectively worse response. Analysis is absent, incoherent, or contradicts the generated rubrics.

Score 2 - Poor:
The rubric fails to capture key differences. Analysis is shallow, rubric questions are too vague to be actionable (e.g., "Is the response good?") or non-discriminative (would yield identical answers for both A and B).

Score 3 - Acceptable:
The rubric identifies some valid differences but misses important ones. Analysis follows the structure but lacks depth. Rubric questions are answerable but either too generic or only partially aligned with the core quality gap between responses.

Score 4 - Good:
The rubric effectively highlights the main quality differences. Analysis is thorough and logically derives the criteria. Rubric questions are specific, discriminative, and appropriately weighted toward what matters most for the user's intent. Minor gaps or suboptimal criterion selection.

Score 5 - Excellent:
The rubric precisely targets the decisive quality differences. Analysis demonstrates clear reasoning through all steps with accurate characterization of both responses. Rubric questions are highly specific, directly answerable, and would reliably guide any evaluator to the correct quality judgment. Criteria selection is optimal.

Output Format:
<evaluation>
[Brief justification highlighting: (1) analysis quality, (2) rubric specificity/discriminativeness, (3) alignment with actual response differences]
</evaluation>
<score>[1-5]</score>

—

[User Question]
{question}

[Assistant A's Answer]
{response_a}

[Assistant B's Answer]
{response_b}

[Generated Rubric]
{rubric}

Figure 10: Prompt template for rubric quality evaluation using GPT-5.

Prompt: Implement a program to find the common elements in two arrays without using any extra data structures.

High-Quality Rubric (Score: 4)	**Low-Quality Rubric (Score: 2)**
<analyze> **Intent:** The user wants a program to find common elements in two arrays **without using any extra data structures**. The core intent is to solve the problem efficiently while adhering to the constraint of no additional memory allocation. **Ideal Answer:** A model answer should use a method that does not require extra memory (e.g., nested loops for unsorted arrays or two-pointer technique for sorted arrays), avoid using lists, sets, or dictionaries to store intermediate results, and provide a working code example. </analyze> <criteria_1> Instruction-following </criteria_1> <rubric_1> Does the response avoid using any extra data structures (e.g., lists, sets, dictionaries) to store intermediate or final results, strictly adhering to the "no extra data structures" constraint? </rubric_1> <criteria_2> Helpfulness </criteria_2> <rubric_2> Does the response provide a working code example that correctly identifies common elements while following the constraint? </rubric_2> <criteria_3> Completeness </criteria_3> <rubric_3> Does the response include all necessary details (e.g., algorithm logic, code, explanation) to solve the problem? </rubric_3>	<analyze> The user's intent is to implement a program that finds common elements without extra data structures. Assistant B offers a nested loop implementation but uses a list to store results, which might violate the constraint. However, the result list is necessary for output, so it may be acceptable. </analyze> <criteria_1> Helpfulness </criteria_1> <rubric_1> Does the response provide a working implementation that correctly identifies common elements? </rubric_1> <criteria_2> Completeness </criteria_2> <rubric_2> Does the response include all necessary code, including the use of a result list to store common elements? </rubric_2> <criteria_3> Instruction-following </criteria_3> <rubric_3> Does the response avoid using structures like lists for storing results, even if they are required for output? </rubric_3>

Figure 11: High-quality vs. low-quality rubrics for implementing common element detection without extra data structures (Qwen3-8B). The high-quality rubric clearly defines the "no extra data structures" constraint and derives consistent evaluation criteria from it. The low-quality rubric shows ambiguous interpretation of the constraint, stating that violating it "may be acceptable," leading to contradictory criteria that simultaneously require and forbid using a result list.

Prompt: I've put a plate on top of an apple, and take the plate from kitchen to dinning room. Where is the apple now

High-Quality Rubric (Score: 4)	**Low-Quality Rubric (Score: 1)**
<analyze> **Intent:** The user wants to know the location of the apple after it has been moved along with a plate from the kitchen to the dining room. **Ideal Answer:** A model answer should clearly state that the apple remains in the kitchen while the plate is moved to the dining room. The apple's location is dictated by the plate's position when it was initially placed on top of it, rather than by subsequent movements of the plate. </analyze> <criteria_1> Completeness </criteria_1> <rubric_1> Does the response accurately describe the apple's final location and mention that the apple did not move with the plate? </rubric_1> <criteria_2> Safety </criteria_2> <rubric_2> Does the response avoid any content that may imply the apple is in the dining room or mislead the user? </rubric_2> <criteria_3> Helpfulness </criteria_3> <rubric_3> Does the response provide a clear explanation that directly addresses the user's question about the apple's location? </rubric_3>	<analyze> **Intent:** The user is asking about the location of the apple in relation to the plate after moving the plate. **Ideal Answer:** A model answer should specify that the apple remains in the kitchen, directly under the plate, since the apple was concealed underneath the plate and did not move. </analyze> <criteria_1> Helpfulness </criteria_1> <rubric_1> Does the response accurately explain the final location of the apple? </rubric_1> <criteria_2> Completeness </criteria_2> <rubric_2> Does the response provide a full explanation including initial and final positions of both objects? </rubric_2> <criteria_3> Safety </criteria_3> <rubric_3> Does the response avoid introducing any dangerous, harmful, or false information about relative positions? </rubric_3>

Figure 12: High-quality vs. low-quality rubrics for a spatial reasoning question (Tulu3-8B-SFT). The high-quality rubric correctly interprets "plate on top of apple" and reasons that moving the plate leaves the apple stationary in the kitchen. The low-quality rubric misunderstands the physical configuration (claiming the apple is "concealed underneath the plate"), and inappropriately applies the Safety criterion to object positions rather than harmful content.

Prompt: Let k and m be real numbers, and suppose that the roots of the equation $x^3 - 7x^2 + kx - m = 0$ are three distinct positive integers. Compute $k + m$.

Assistant A (Chosen Response):

Given the polynomial $x^3 - 7x^2 + kx - m = 0$ with roots a, b, c as distinct positive integers, by Vieta's formulas: $a + b + c = 7$, $ab + bc + ca = k$, $abc = m$. Since a, b, c are distinct positive integers summing to 7, the only solution is $\{1, 2, 4\}$. Thus: $k = 1 \cdot 2 + 2 \cdot 4 + 4 \cdot 1 = 14$, $m = 1 \cdot 2 \cdot 4 = 8$. Therefore, $k + m = 14 + 8 = \mathbf{22}$.

Assistant B (Rejected Response):

Given the polynomial with roots as distinct positive integers, by Vieta's formulas: $a + b + c = 7$. The smallest distinct positive integers are $1, 2, 3$. Using these: $k = 1 \cdot 2 + 2 \cdot 3 + 3 \cdot 1 = 11$, $m = 1 \cdot 2 \cdot 3 = 6$. Therefore, $k + m = 11 + 6 = \mathbf{17}$.

Base Model (Tulu3-8B-SFT, Score: 2)

<analyze>
Intent: The User Question seeks to find the values of k and m in the polynomial equation given that the roots are three distinct positive integers. The goal is to compute $k + m$.
Comparison: Assistant A correctly identifies the roots as 1, 2, and 4 and computes $k + m$ as 22. Assistant B correctly identifies the roots as 1, 2, and 3 and computes $k + m$ as 17. Assistant A's answer is more accurate because it correctly matches the polynomial's roots with the given constraints.
</analyze>
<criteria_1> Completeness </criteria_1> <rubric_1> Does the answer include all necessary steps to derive the values of k and m? </rubric_1>
<criteria_2> Helpfulness </criteria_2> <rubric_2> Does the answer directly address the user's intent and provide a correct and relevant solution? </rubric_2>

C2 Generator (Tulu3-8B-SFT + DPO, Score: 4)

<analyze>
Intent: The core point of the user question is to find the values of k and m such that the roots of the polynomial are three distinct positive integers, and then compute $k + m$.
Ideal Answer: The ideal answer should correctly identify the roots that are distinct positive integers summing to 7, use Vieta's formulas to calculate k and m, and compute $k + m$ accurately.
</analyze>
<criteria_1> Helpfulness </criteria_1> <rubric_1> Does the answer directly address the user's question of finding k and m and computing $k + m$? </rubric_1>
<criteria_2> Completeness </criteria_2> <rubric_2> Does the answer correctly identify the roots that satisfy the condition $a + b + c = 7$ and use Vieta's formulas to compute k and m? </rubric_2>
<criteria_3> Accuracy in calculations </criteria_3> <rubric_3> Are the values of k and m calculated accurately, and does the sum $k + m$ match the correct answer? </rubric_3>

Larger Model (Tulu3-70B, Score: 5)

<analyze>
Intent: The core point of the User Question is to determine the values of k and m given that the roots of the polynomial are three distinct positive integers and then compute $k + m$.
Ideal Answer: The ideal answer should correctly identify the roots as 1, 2, and 4, calculate k and m based on these roots, and then compute $k + m$.
</analyze>
<criteria_1> Helpfulness </criteria_1> <rubric_1> Does the answer correctly identify the roots of the polynomial as distinct positive integers that sum to 7? </rubric_1>
<criteria_2> Completeness </criteria_2> <rubric_2> Does the answer correctly compute k and m based on the identified roots? </rubric_2>
<criteria_3> Instruction-following </criteria_3> <rubric_3> Does the answer accurately compute $k + m$? </rubric_3>

Figure 13: Rubric comparison across Tulu3 family models. The base model produces generic rubrics ("Completeness" and "Helpfulness") that fail to target the critical constraint: roots must sum to 7. The C2 generator explicitly identifies this constraint in its criteria, enabling accurate discrimination between correct (roots 1, 2, 4) and incorrect (roots 1, 2, 3) solutions.

Prompt: The greatest common divisor of positive integers m and n is 8. The least common multiple of m and n is 112. What is the least possible value of $m + n$?

Assistant A (Chosen Response):

Using $\gcd(m, n) \cdot \mathrm{lcm}(m, n) = m \cdot n$, we get $8 \cdot 112 = 896$, so $m \cdot n = 896$. Let $m = 8a$ and $n = 8b$ where $\gcd(a, b) = 1$. Then $64ab = 896$, giving $ab = 14$. The coprime pairs (a, b) with product 14 are: $(1, 14)$ and $(2, 7)$. For $(1, 14)$: $m + n = 8 + 112 = 120$. For $(2, 7)$: $m + n = 16 + 56 = 72$. The minimum is $\mathbf{72}$.

Assistant B (Rejected Response):

Using $\gcd(m, n) \cdot \mathrm{lcm}(m, n) = m \cdot n$, we get $m \cdot n = 896$. Let $m = 8a$ and $n = 8b$ where $\gcd(a, b) = 1$. Then $ab = 14$. The pairs (a, b) satisfying this are $(1, 14)$ and $(14, 1)$. For both cases: $m + n = 8 + 112 = 120$. Therefore, the minimum is $\mathbf{120}$.

Base Model (Qwen3-8B, Score: 3)

<analyze>
Intent: The user wants the minimal sum of two numbers with specific GCD and LCM. The key here is to use the relationship between GCD and LCM.
Ideal Answer: Must enumerate all coprime pairs of a and b that multiply to 14, compute $m + n$ for each pair and select the minimum.
</analyze>
<criteria_1> Completeness </criteria_1> <rubric_1> Does the answer include all valid coprime pairs (a, b) such that $ab = 14$, including $(1, 14), (2, 7), (7, 2), (14, 1)$? </rubric_1>
<criteria_2> Helpfulness </criteria_2> <rubric_2> Does the answer correctly identify the minimal possible value of $m + n$ as 72, or does it provide an incorrect value (e.g., 120)? </rubric_2>
<criteria_3> Instruction-following </criteria_3> <rubric_3> Does the answer adhere to the mathematical principles and logical steps required to solve the problem? </rubric_3>

C2 Generator (Qwen3-8B + DPO, Score: 4)

<analyze>
Intent: The user seeks the minimal sum of two integers m and n given their GCD is 8 and LCM is 112.
Ideal Answer: A complete answer must use the formula GCD $\times$ LCM $= m \times n$, factor $m = 8a$, $n = 8b$, enumerate all coprime pairs, and select the minimum.
</analyze>
<criteria_1> Completeness </criteria_1> <rubric_1> Does the answer include all valid coprime factor pairs of $ab = 14$ (e.g., $(1, 14)$ and $(2, 7)$) and compute the corresponding $m + n$ values? </rubric_1>
<criteria_2> Helpfulness </criteria_2> <rubric_2> Does the answer correctly identify the least possible value of $m + n$ as 72, or does it provide an incorrect minimal sum (e.g., 120)? </rubric_2>

Larger Model (Qwen3-32B, Score: 4)

<analyze>
Intent: The user wants to find the minimal sum of two positive integers m and n given their GCD is 8 and LCM is 112.
Ideal Answer: Must correctly enumerate all coprime pairs and select the minimum sum.
</analyze>
<criteria_1> Completeness </criteria_1> <rubric_1> Does the answer enumerate all coprime integer pairs (a, b) such that $ab = 14$? </rubric_1>
<criteria_2> Helpfulness </criteria_2> <rubric_2> Does the answer correctly identify the minimum possible value of $m + n$ based on the derived pairs? </rubric_2>
<criteria_3> Instruction-following </criteria_3> <rubric_3> Does the answer adhere to the mathematical process of factoring m and n as multiples of their GCD? </rubric_3>

Figure 14: Rubric comparison across Qwen3 family models. All three models correctly identify the key issue—the missing $(2, 7)$ coprime pair—but differ in rubric specificity. The C2 generator produces the most focused criteria by directly asking whether all coprime pairs are enumerated and whether the correct minimum (72) is identified, without redundant criteria.

Case 1: Correctly Following a Helpful Rubric	*(Qwen3-8B)*

Prompt: How can I make bubble solution?

Response A:
Create bubble solution by mixing water, vinegar, and optional glycerin or corn syrup, then use a homemade wand to blow bubbles for a fun activity.

Response B:
Making your own bubble solution at home is a fun and easy activity for kids and adults alike. Here are a few simple recipes to get you started. Basic Bubble Solution Recipe. Ingredients: 6 cups of water, 1/2 cup of dish soap (like Dawn or Joy), 2 tablespoons of glycerin (optional, but helps create longer-lasting bubbles). Instructions: Mix the water and dish soap in a large container. If using glycerin, add it to the mixture and stir well. Let the solution sit for a few hours or overnight to allow the soap to dissolve fully and the bubbles to become more stable. Use a bubble wand or a homemade wand (like a wire hanger bent into a circle) to blow bubbles. Advanced Bubble Solution Recipe. Ingredients: 4 cups of water, 1/2 cup of dish soap, 1/4 cup of corn syrup (helps create larger bubbles), 1 tablespoon of baking powder (optional, for extra strength). Instructions: Combine the water, dish soap, and corn syrup in a large container. If using baking powder, add it to the mixture and stir until dissolved. Let the solution sit for a few hours or overnight. Use a bubble wand to blow bubbles. Homemade Bubble Wands. If you don't have a bubble wand, you can easily make one. Wire Hanger Wand: Straighten a wire hanger and bend one end into a circle. Twist the ends together to secure the circle. Yarn Wand: Wrap yarn around a wire hanger or a stick to create a fluffy wand that can catch more solution. Plastic Ring Wand: Use the ring from a milk jug or any other plastic ring. Dip it into the solution and blow through the center. Tips for Blowing Bubbles. Warm Solution: Warm water can help create better bubbles. Gentle Blowing: Blow gently to create larger, longer-lasting bubbles. Outdoor Play: Bubbles are best enjoyed outdoors where there is less chance of them popping quickly due to dust or drafts. Enjoy your bubble-blowing adventure!

Generated Rubric:
The user is asking for a method to create bubble solution, likely for a fun activity. The core intent is to provide a clear, actionable recipe with ingredients and steps. A good answer should be practical, safe, and detailed enough for someone to replicate successfully.
The ideal answer would include: 1. Clear ingredients (e.g., water, dish soap, optional additives like glycerin/corn syrup). 2. Step-by-step instructions. 3. Optional variations or advanced techniques. 4. Safety notes (e.g., avoiding harmful substances).
Assistant B excels in Completeness and Helpfulness, covering all necessary aspects with accurate information. Assistant A is incomplete, lacks detail, and includes an incorrect ingredient (vinegar).
The key differentiators are Completeness (detail and variety) and Helpfulness (accuracy and practicality). Instruction-following is not applicable here since there are no formatting constraints. Safety is secondary but relevant for avoiding harmful ingredients.
<criteria_1> Completeness </criteria_1> <rubric_1> Does the response include multiple detailed recipes with precise measurements, ingredients, and step-by-step instructions for making bubble solution, along with optional variations and practical tips? </rubric_1>
<criteria_2> Helpfulness </criteria_2> <rubric_2> Is the explanation accurate and practical, avoiding misleading ingredients (e.g., vinegar) and providing actionable steps that allow the user to successfully create bubble solution? </rubric_2>
<criteria_3> Safety </criteria_3> <rubric_3> Does the response avoid recommending potentially harmful ingredients or unsafe practices (e.g., using non-bubble-friendly substances)? </rubric_3>

Verifier Reasoning:
Okay, let's start by understanding the user's question: "How can I make bubble solution?" The core intent here is to provide a method for creating bubble solution, likely for a fun activity, possibly for kids. The user probably wants a simple, safe, and effective recipe.
Now, looking at the two assistants' responses. Assistant A gives a very brief answer: water, vinegar, glycerin or corn syrup, and a homemade wand. That's minimal. Assistant B provides two detailed recipes with precise measurements, ingredients, step-by-step instructions, homemade wand ideas, and tips.
The provided rubric has three criteria: Completeness, Helpfulness, and Safety. Let's check each.
Completeness: Assistant B includes multiple recipes (basic and advanced), detailed ingredients, steps, and tips. Assistant A is too vague and lacks specifics. So B is more complete.
Helpfulness: Assistant B's answer is accurate, avoiding vinegar (which might not be standard and could be harmful). Assistant A includes vinegar, which might not be ideal. So B is more helpful.
Safety: Assistant B doesn't mention any harmful ingredients, while Assistant A's use of vinegar could be a safety concern if not properly handled. So B is safer.
Now, evaluating the rubric's validity. The rubric's criteria align with the user's intent. Completeness, Helpfulness, and Safety are all relevant. The rubric correctly identifies that Assistant B provides more detailed and accurate information, while Assistant A is incomplete and includes a potentially misleading ingredient. The rubric seems reliable because it addresses the user's needs accurately. Therefore, the rubric is reliable, and I should use it as is.
Comparing the two assistants, Assistant B clearly meets all the criteria better. So the answer should be B.

Assessment: <rubric>helpful</rubric> **Judgment:** <answer>B</answer> (*Correct*)

Figure 15: The verifier correctly identifies a helpful rubric that highlights the accuracy criterion (avoiding vinegar as a misleading ingredient). By following this rubric, the verifier selects Response B, which provides correct ingredients and detailed instructions, over Response A, which incorrectly includes vinegar.

Case 2: Correctly Rejecting a Misleading Rubric *(Tulu3-8B-SFT)*

Prompt: Write a C++ function `vector<string> words_string(string s)` to solve the following problem: You will be given a string of words separated by commas or spaces. Your task is to split the string into words and return a vector of the words. For example: `words_string("Hi, my name is John") == {"Hi", "my", "name", "is", "John"}`

Response A:
To solve the problem of splitting a string into words separated by commas or spaces, you can use the following C++ code:

```
#include <iostream> #include <vector> #include <string> #include <sstream>
using namespace std;
vector<string> words_string(string s) {    vector<string> words;    stringstream ss(s);    string word;
    while (getline(ss, word, ',')) {        stringstream word_ss(word);        string token; while (word_ss » token) {            words.push_back(token);        }    }
    return words; }
```

This function uses a two-step process to split the string: first by commas, then by spaces within the substrings.

Response B:
To solve the problem of splitting a string into words separated by commas or spaces, we need to implement a function that can handle both delimiters. Here's a complete implementation in C++:

```
#include <vector> #include <string> #include <sstream> #include <algorithm>
using namespace std;
vector<string> words_string(string s) {    vector<string> words;    stringstream ss(s);    string word;
    while (ss » word) {        word.erase(remove(word.begin(), word.end(), ','), word.end()); words.push_back(word);    }
    return words; }
```

This function extracts words from the stream one by one and removes any commas from the extracted word before adding it to the vector.

Rubric:
The user is asking for a C++ function that splits a string into words, handling both commas and spaces as delimiters. The core intent is to correctly parse the input string into a vector of strings, where each string represents a word.
Ideal Answer: The ideal answer should efficiently split the input string into words, handling both commas and spaces as delimiters. It should correctly parse the input string into a vector of strings, where each string represents a word, and handle cases where words contain commas.
Comparison: - Assistant A correctly handles both commas and spaces as delimiters but does so in a more complex manner, which might be considered less efficient. - Assistant B simplifies the process, making it more straightforward and potentially more efficient. However, it might not handle cases where a word contains a comma as intended, as it removes commas from the words.
`<criteria_1>` Helpfulness `</criteria_1>` `<rubric_1>` Does the function correctly split the input string into words, handling both commas and spaces as delimiters? `</rubric_1>` `<criteria_2>` Completeness `</criteria_2>` `<rubric_2>` Does the function correctly handle words that contain commas, ensuring they are not removed or split incorrectly? `</rubric_2>`

Verifier Reasoning:
First, let's address the Ideal Answer: - It should correctly handle both commas and spaces as delimiters. - It should parse the input string into a vector of strings, where each string represents a word. - It should handle cases where words contain commas without removing or splitting them incorrectly.
Now, let's analyze the provided rubric and Ideal Answer:
- Helpfulness: This is a crucial aspect of the Ideal Answer. Both Assistant A and B should correctly split the input string into words, handling both commas and spaces as delimiters. Assistant A does this by using a two-step process to first split by commas and then by spaces within the substrings, which correctly handles both delimiters. Assistant B simplifies the process by using a single loop to extract words from the string stream, but it removes commas from each word before adding it to the vector. This could lead to incorrect handling of words with commas.
- Completeness: The rubric doesn't directly address this aspect, but the Ideal Answer emphasizes that the function should handle cases where words contain commas. Assistant A correctly handles this by splitting the input string into words with both delimiters. Assistant B, however, removes commas from the words, which can be incorrect for words with commas.
Given the analysis, we can conclude the following: - The ideal solution should be helpful, handling both delimiters correctly. - Completeness and efficiency are also important, but Assistant A seems to be more comprehensive in handling both delimiters, while Assistant B simplifies the process but may not handle words with commas correctly.
The rubric's criterion about "words containing commas" misinterprets the problem—commas are delimiters in this task, not characters within words. Both solutions correctly treat commas as separators.

Assessment: `<rubric>misleading</rubric>` **Judgment:** `<answer>A</answer>` (*Correct*)

Figure 16: The verifier correctly identifies that the rubric misinterprets the problem requirements by treating commas as possible word characters rather than delimiters. By rejecting the misleading criterion and falling back to evaluating delimiter handling, the verifier makes the correct judgment.

Case 3: Incorrectly Rejecting a Helpful Rubric	*(Tulu3-8B-SFT)*

Prompt: I live in Scotland and would like to know the most cost effective way to heat my house considering the rise in both gas and electric prices. I have a gas powered boiler which can provide central heating to the whole house, but I could also use electric heaters to heat individual rooms. Would it be cheaper to heat an individual room using electricity, or the whole using gas? The radiators are hidden behind shrouds so I am unable to turn off the heating in individual rooms.

Response A:
Heating individual rooms with electric heaters may be more cost-effective and environmentally friendly than using a gas boiler to heat the entire house, especially given the rise in energy prices and the inability to control individual radiators.

Response B:
Given the current rise in both gas and electric prices, it's important to consider the efficiency and cost of both heating methods. Here are some factors to consider: First, Energy Efficiency: Modern gas boilers are generally very efficient, often operating at around 90-94% efficiency. This means that for every unit of gas used, you get close to 1 unit of heat energy. Electric heaters are 100% efficient because all the electricity consumed is converted directly into heat. However, electricity is typically more expensive per unit than gas. Second, Cost Comparison: The cost of gas is generally lower per unit of energy compared to electricity. However, the exact cost will depend on your energy supplier and the current market prices. Electricity is more expensive per unit of energy, but if you only need to heat a single room, the cost might be lower than heating the entire house with gas. Third, Heating Individual Rooms vs. Whole House: If you use your gas boiler to heat the entire house, you will be paying for the gas used to heat all the radiators, even if some rooms are not occupied. This can be inefficient if you only need to heat a few rooms. Using electric heaters to heat individual rooms can be more cost-effective if you only need to heat a specific area. You can turn off the gas central heating and use electric heaters in the rooms you are using. Fourth, Practical Considerations: Since your radiators are hidden behind shrouds, you can't easily control the heating in individual rooms. This means you might end up heating rooms you don't need to, which is inefficient. If possible, consider installing Thermostatic Radiator Valves (TRVs) on your radiators. These valves allow you to control the temperature in each room individually, making it easier to manage heating costs. Fifth, Alternative Solutions: Consider using a smart thermostat that allows you to control the heating remotely and set different temperatures for different times of the day. This can help you optimize your heating usage.

Rubric:
The user's intent is to find the most cost-effective way to heat their house in Scotland, considering the rise in gas and electric prices. They have a gas-powered boiler for central heating but are considering using electric heaters for individual rooms due to the inability to control the heating in individual rooms with the current setup.
The ideal answer should provide a comprehensive comparison of the costs and efficiencies of both heating methods, taking into account the specific circumstances of the user. It should also offer practical solutions to improve the heating system's efficiency.
Criteria and Rubrics: 1. Completeness: Does the answer provide a detailed comparison of the costs and efficiencies of both heating methods? 2. Helpfulness: Does the answer offer practical solutions to improve the heating system's efficiency? 3. Safety: Does the answer avoid promoting unsafe or harmful practices?

Verifier Reasoning:
Analyzing the user's question, the core intent is to determine the most cost-effective way to heat their house given the rising costs of gas and electricity, and they want to compare heating individual rooms with electric heaters vs. using a gas boiler for the whole house. The ideal answer would provide a clear cost comparison, mention energy efficiency of both methods, and address the practical limitation of not being able to turn off the heating in individual rooms due to radiator shrouds.
The provided rubric does not specifically mention a need for a comparison of energy efficiency between electric heaters and gas boilers, nor does it address the practical limitation regarding the radiator shrouds. The rubric only mentions the user's intent to compare costs, which does not capture all the considerations the user mentioned. Therefore, the rubric is unreliable and should be disregarded.
Given that the rubric is unreliable, let's define our own rubric: - Rubric for Helpfulness: Does the response provide a clear, cost-efficient comparison of heating the house with gas or using electric heaters for individual rooms, considering the constraint of not being able to turn off radiators individually? - Rubric for Completeness: Does the response cover all considerations needed for a proper comparison, including the energy efficiency of both methods and the practical limitation of not being able to control individual room heating due to radiator shrouds? - Rubric for Safety: Does the response avoid promoting any unsafe practices or techniques and is it adherent to ethical guidelines? - Rubric for Instruction-following: Does the response follow the instructions provided in the question, without additional length or length modifications?

Assessment: `<rubric>misleading</rubric>` **Judgment:** `<answer>A</answer>` (*Incorrect—should be B*)

Figure 17: The verifier incorrectly rejects a helpful rubric that appropriately emphasizes cost comparison and practical solutions. Despite the rubric correctly identifying Response B as more comprehensive, the verifier dismisses it and selects the brief, unsupported Response A.

Case 4: Incorrectly Trusting a Misleading Rubric *(Qwen3-8B)*

Prompt:
Write a C++ function vector<int> make_a_pile(int n) to solve the following problem: Given a positive integer n, you have to make a pile of n levels of stones. The first level has n stones. The number of stones in the next level is: - the next odd number if n is odd. - the next even number if n is even. Return the number of stones in each level in a vector... Examples: make_a_pile(3) → {3, 5, 7}

Response A:
To solve this problem, we need to create a function make_a_pile... Here's the implementation:

```
vector<int> make_a_pile(int n) {
    vector<int> pile; int current = n;
    while (current > 0) {
        pile.push_back(current);
        if (current % 2 == 0) { current += 2; } else { current += 2; }
    }
    return pile;
}
```

... (Explanation and main function omitted) ...

Response B:
To solve the problem, we need to follow the rules provided... Here's the step-by-step approach:

```
vector<int> make_a_pile(int n) {
    vector<int> result; int current_stones = n;
    for (int i = 0; i < n; ++i) {
        result.push_back(current_stones);
        if (current_stones % 2 == 0) { current_stones += 2; }
        else { current_stones += 2; }
    }
    return result;
}
```

... (Explanation and main function omitted) ...

Rubric:
Intent: The core intent is to generate a sequence of n numbers starting from n and incrementing by 2 each time, ensuring the function runs exactly n times to create the correct number of levels.
Ideal Answer: The ideal answer should utilize a loop that iterates exactly n times (e.g., for loop) to generate the vector elements, ensuring termination and correctness.
<criteria_1> Helpfulness </criteria_1> <rubric_1> Does the function correctly generate n elements, ensuring the loop runs exactly n times (e.g., using a counter or for-loop) rather than relying on a condition that might lead to an infinite loop (e.g., while(current > 0))? </rubric_1>
<criteria_2> Instruction-following </criteria_2> <rubric_2> Does the code adhere to the requirement of returning a vector where element at index i represents level i+1, maintaining the correct parity (odd/even)? </rubric_2>

Verifier Reasoning:
Alright, let's break this down step by step.
First, I need to understand the user's question. They want a C++ function called make_a_pile(int n) that returns a vector of integers. The first level has n stones, and each subsequent level increases by 2, maintaining the parity (odd or even) of the initial n. The examples given show that for n=3, the output is {3,5,7}.
Now, looking at the two assistants' answers.
Assistant A uses a for loop that runs exactly n times. They start with current_stones = n and add 2 each time, regardless of whether the current value is even or odd. This correctly generates the sequence as required. The loop ensures exactly n elements are added, which matches the problem's requirement.
Assistant B uses a while loop with the condition current > 0. They start with current = n and add 2 each time. However, since current keeps increasing by 2, the loop will never terminate. For example, if n=3, the loop will add 3, then 5, then 7, and so on infinitely. This is a critical error because the function is supposed to return exactly n elements, not an infinite sequence.
The rubric provided by the user emphasizes correctness in loop control and parity handling. The key points are ensuring the loop runs exactly n times and that the parity of the initial n is maintained. Assistant A's code correctly uses a for loop with n iterations, while Assistant B's code has an infinite loop due to the while condition.
Additionally, both assistants correctly handle the parity by adding 2 each time, which maintains the initial parity. However, the loop termination is the main issue here.
So, based on the rubric, Assistant A's answer is correct and follows the instructions properly, while Assistant B's answer has a critical flaw in the loop condition leading to an infinite loop. Therefore, Assistant A's response is superior.

Assessment: <rubric>helpful</rubric> **Judgment:** <answer>A</answer> (*Incorrect—The Verifier hallucinated the models' contents, swapping A and B*)

Figure 18: An example of a verification failure caused by model hallucination. Although the generated rubric correctly analyzed the code (identifying Assistant A as having the infinite loop), the verifier hallucinated the content of the responses: it explicitly stated "Assistant A uses a for loop" and "Assistant B uses a while loop," effectively swapping the models. This led to the selection of the incorrect response despite the rubric's accurate guidance.