CO-EVOLVING LLM EVALUATORS AND POLICIES VIA DYNAMICRUBRIC

Anonymous authors
Paper under double-blind review

ABSTRACT

Post-training with evaluator feedback on policy-induced samples serves as a major mechanism for improving large language models. As policies improve, these sampled responses become close in quality. These close responses create a bottleneck for policy optimization: when evaluator score gaps collapse or reverse, policy supervision becomes weak or misleading. We theoretically characterize why these gaps matter through a probability allocation view: shifting probability mass from one response to another yields a directional gain exactly equal to the evaluator score gap between them. This identifies relative score gaps as the policy optimization signals that guide updates. We therefore propose **DynamicRubric** to co-evolve evaluators and policies using response-set-conditioned rubrics. An LLM can take either role and self-evolve by learning to supervise its own updates without an external reward model or judge. Using an 8B backbone, DynamicRubric produces an evaluator that is competitive with a 70B reward model on preference evaluation. On open-ended tasks, DynamicRubric provides stronger policy supervision than static rubrics generated by a 235B model, and performance gains generalize to verifiable tasks. A DynamicRubric-optimized model is fully deployed for AI answering in a large-scale production environment, handling tens of millions of requests per day and improving key online metrics. These results suggest a principle for post-training: evaluators should evolve with the policies they supervise.

1 INTRODUCTION

Post-training with evaluator feedback has become central to improving large language models (LLMs) (Ouyang et al., 2022; Rafailov et al., 2023). In these pipelines, policies are optimized using feedback on responses sampled from the current policy. As optimization improves a policy, its sampled responses often become close in quality, making the remaining useful differences subtle and instance-specific. This creates a central bottleneck for evaluator-guided optimization: policy updates depend on relative score gaps among close candidates, which determine how probability mass should be reallocated. Existing reward models and LLM-as-Judge evaluators are often unreliable for such fine-grained comparisons (Lambert et al., 2024; Tan et al., 2025). When these gaps collapse or point in the wrong direction, policy optimization receives weak or misleading feedback.

Rubric-based evaluation offers a natural way to make fine-grained differences explicit by decomposing holistic judgments into criteria (Liu et al., 2023; Ye et al., 2023; Kim et al., 2024; Song et al., 2024). Yet a rubric helps policy optimization only when its criteria separate the responses that the current policy actually produces. As policies improve, these candidate responses often satisfy broad criteria such as helpfulness and clarity. Once these broad criteria become saturated, the distinctions that matter for policy optimization become more instance-specific: a missed constraint, a flawed reasoning step, or a choice that fails to match the user's intent. Rubrics written before observing the candidate responses can easily overlook the distinctions that determine the next policy update.

Taken together, this gap-preservation requirement motivates *dynamic*, response-set-conditioned rubric evaluation. Given a prompt and responses sampled from the current policy, the evaluator generates binary rubric items aimed at the differences that matter within the set. A verifier applies these items to every response and aggregates the yes/no judgments into response-level scores. The shared rubric makes the resulting score gaps directly comparable within the set and traceable to explicit judgments, yielding feedback aligned with the comparisons that drive policy optimization.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

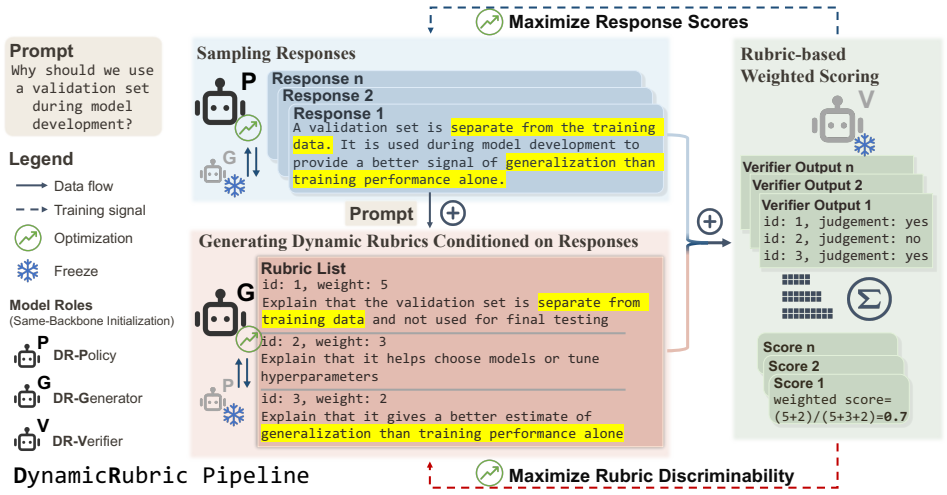


Figure 1: Overview of DynamicRubric. For each prompt, DR-Policy samples a candidate response set. DR-Generator then produces weighted binary rubric items conditioned on the candidate responses. A frozen DR-Verifier applies the rubrics to produce response-level scores for co-evolution.

We theoretically ground this design through a probability allocation view of policy optimization. Under this view, conditioned on a prompt and a sampled response set, the directional gain of shifting probability mass from one candidate to another is exactly the evaluator score gap between them. This identity turns evaluator discriminability into an optimization quantity: relative score gaps are the signals that guide policy updates within the response set. We further establish that response-set-conditioned evaluators include prompt-only evaluators as a special case, while the response-set input allows the evaluator to adapt to the candidates being compared. Since policy updates continually change the response distribution, the evaluator must adapt to the response sets it is used to supervise. Together, these results yield an evaluator–policy co-evolution principle: adapt the evaluator to current-policy response sets, then optimize the policy using the adapted evaluator.

To instantiate this bilevel principle, we propose **DynamicRubric**, a response-set-conditioned rubric framework illustrated in Figure 1. Given a prompt and current-policy candidate responses, **DR-Generator** produces weighted binary rubric items targeting distinctions in the set. **DR-Verifier** applies each item to every response and aggregates weighted judgments into response-level scores. DynamicRubric alternates between adapting DR-Generator to response sets from the evolving **DR-Policy** and optimizing DR-Policy with these scores. The updated policy supplies next response sets. DynamicRubric lets copies of one LLM play these roles, thereby realizing a self-evolution paradigm.

Experiments show that DynamicRubric improves both evaluator performance and downstream policy optimization. Using Qwen3-8B as the backbone, the trained DR-Generator outperforms larger evaluator baselines on preference evaluation, including a Qwen3-32B dynamic rubric generator. When used as the supervision signal for policy optimization, DynamicRubric consistently improves open-ended generation over supervision from a 70B reward model and static rubric supervision from a 235B rubric generator. Policies optimized with DynamicRubric consistently outperform the initial policy on math, coding, and scientific QA benchmarks. These results show that DynamicRubric's gains in policy performance extend beyond open-ended generation to verifiable tasks. Continuous co-evolution experiments further show that evolved evaluators provide stronger supervision than stale evaluators. Beyond benchmark evaluation, we deploy a DynamicRubric-optimized model in a large-scale online production AI answering setting that serves tens of millions of requests per day. Online A/B tests against the previous production model show statistically significant improvements in total search volume, user duration, and absolute positive user behaviors. The resulting model is fully deployed in production and currently serves all online traffic, replacing the previous production model. Together, these results point to a central requirement for post-training evaluators: they should evolve with the policies to maintain relative discriminability. DynamicRubric meets this requirement by letting the rubric evaluator evolve with the response sets encountered during optimization, keeping evaluation focused on the comparisons that determine the next policy update.

2 Related Work

2.1 Post-training with Feedback

Post-training with feedback is central to improving LLMs. RLHF learns reward models from human preferences, while RLAIF replaces part of this supervision with AI feedback (Ouyang et al., 2022; Bai et al., 2022). Direct preference optimization trains policies from preference pairs without a separate reward model (Rafailov et al., 2023). Recent reasoning systems use verifiable outcome rewards for mathematical, coding, and general reasoning tasks (Shao et al., 2024; Guo et al., 2025a). Across evaluator-guided methods, policy improvement depends on reliable distinctions among current-policy candidates, while imperfect rewards can cause overoptimization (Gao et al., 2023). This motivates feedback that remains discriminative as policy responses shift.

2.2 Rubric-based Evaluation for LLM Optimization

LLM evaluators are widely used for open-ended generation and preference assessment (Zheng et al., 2023; Liu et al., 2023). Rubric-based methods decompose response quality into explicit, interpretable criteria. G-Eval and FLASK structure evaluation around forms or fine-grained alignment skills, while Prometheus and FineSurE support customized or multidimensional evaluation (Liu et al., 2023; Ye et al., 2023; Kim et al., 2024; Song et al., 2024). Rubrics also enable post-training beyond domains with deterministic verifiers (Gunjal et al., 2025; Zhang et al., 2025). Recent work jointly evolves rubrics and policies in open-ended reinforcement learning (Guan et al., 2026; Ding et al., 2026). However, how to keep such criteria discriminative as policy improves remains unexplored.

3 Methodology

3.1 Relative Evaluator Score Gaps as Optimization Signals

In LLM post-training, useful evaluators are expected to maintain discriminability among close current-policy responses (Ouyang et al., 2022; Zhang et al., 2025). This section shows that relative evaluator score gaps within policy-induced response sets are exactly what guides policy updates. Formally, let $x \sim \mathcal{D}$ be a prompt and $\pi_\theta(y \mid x)$ the current policy. Let $E(x, y) \in \mathbb{R}$ be the evaluator's scalar score for (x, y). We sample K responses from $\pi_\theta(\cdot \mid x)$ to form the response set $\mathcal{C}$:

$$y_1, \ldots, y_K \overset{\text{i.i.d.}}{\sim} \pi_\theta(\cdot \mid x), \qquad \mathcal{C} = \{y_1, \ldots, y_K\}. \tag{1}$$

To characterize relative score structure, let $P(\cdot \mid x, \mathcal{C})$ denote an analysis-time distribution over ordered response pairs (y^+, y^-) in $\mathcal{C}$. Evaluator-independent reference quality determines the order. The average **relative evaluator score gap** over this distribution is defined as follows:

$$\Delta_{\text{gap}}(E; \theta) = \mathbb{E}_{x\sim\mathcal{D}}\mathbb{E}_{\mathcal{C}\sim\pi_\theta(\cdot|x)}\mathbb{E}_{(y^+,y^-)\sim P(\cdot|x,\mathcal{C})}\left[E(x, y^+) - E(x, y^-)\right]. \tag{2}$$

Equation 2 measures the average relative score gap over ordered pairs in $\mathcal{C}$. From a local probability allocation perspective, this gap is also the local optimization signal for moving probability mass from y^- to y^+. To see this, for a fixed x and $\mathcal{C}$, we define the normalized policy mass on $\mathcal{C}$ as

$$\alpha_i = \frac{\pi_\theta(y_i \mid x)}{\sum_{j=1}^K \pi_\theta(y_j \mid x)}, \qquad \sum_{i=1}^K \alpha_i = 1. \tag{3}$$

In the local coordinates $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_K)$, we consider the local proxy objective

$$\mathcal{J}^{\text{local}}_{x,\mathcal{C}}(\boldsymbol{\alpha}; E) = \sum_{i=1}^K \alpha_i E(x, y_i). \tag{4}$$

Let $\mathbf{d} = (d_1, \ldots, d_K)$ be any feasible local direction on the probability simplex, satisfying $\sum_{i=1}^K d_i = 0$, and define the local path $\boldsymbol{\alpha}(\varepsilon) = \boldsymbol{\alpha} + \varepsilon\mathbf{d}$. For sufficiently small $|\varepsilon|$, this path remains inside the simplex. Since $\mathcal{J}^{\text{local}}_{x,\mathcal{C}}$ is linear in $\boldsymbol{\alpha}$, its directional derivative along this path is

$$\frac{\mathrm{d}}{\mathrm{d}\varepsilon}\mathcal{J}^{\text{local}}_{x,\mathcal{C}}(\boldsymbol{\alpha} + \varepsilon\mathbf{d}; E)\Big|_{\varepsilon=0} = \sum_{i=1}^K d_i E(x, y_i). \tag{5}$$

Thus, the local improvement directions on $\mathcal{C}$ are determined by the relative structure of evaluator scores. For any pair (y_i, y_j), define the transfer direction $\mathbf{d}_{i\leftarrow j} = e_i - e_j$, where e_i, e_j are standard basis vectors. This direction transfers an infinitesimal amount of probability mass from y_j to y_i while leaving the remaining coordinates unchanged. The directional derivative along this direction is

$$\frac{\mathrm{d}}{\mathrm{d}\varepsilon}\mathcal{J}^{\mathrm{local}}_{x,\mathcal{C}}(\boldsymbol{\alpha} + \varepsilon\mathbf{d}_{i\leftarrow j}; E)\Big|_{\varepsilon=0} = E(x, y_i) - E(x, y_j). \tag{6}$$

Thus, for an ordered pair (y^+, y^-), the relative score gap is exactly the local directional gain of transferring probability mass from y^- to y^+. Since general local updates can be decomposed into such pairwise transfers, relative score gaps characterize the local improvement directions on $\mathcal{C}$. When these gaps collapse, the optimization signal exposed by the evaluator shrinks, and policy optimization becomes less effective on the sampled response set. Equation (6) holds pointwise for each fixed x, $\mathcal{C}$, and ordered pair (y^+, y^-). Taking expectation under the same sampling process gives the expected local **optimization signal**. Let $\mathbf{d}_{+\leftarrow -}$ denote the transfer direction from y^- to y^+. Then

$$\begin{aligned}
&\mathbb{E}_{x\sim\mathcal{D}}\mathbb{E}_{\mathcal{C}\sim\pi_\theta(\cdot|x)}\mathbb{E}_{(y^+,y^-)\sim P(\cdot|x,\mathcal{C})}\left[\frac{\mathrm{d}}{\mathrm{d}\varepsilon}\mathcal{J}^{\mathrm{local}}_{x,\mathcal{C}}(\boldsymbol{\alpha} + \varepsilon\mathbf{d}_{+\leftarrow -}; E)\Big|_{\varepsilon=0}\right] \\
&= \mathbb{E}_{x\sim\mathcal{D}}\mathbb{E}_{\mathcal{C}\sim\pi_\theta(\cdot|x)}\mathbb{E}_{(y^+,y^-)\sim P(\cdot|x,\mathcal{C})}\left[E(x, y^+) - E(x, y^-)\right] \\
&= \Delta_{\mathrm{gap}}(E; \theta) \qquad \text{(by Equation (2))}.
\end{aligned} \tag{7}$$

Therefore, $\Delta_{\mathrm{gap}}(E; \theta)$ is the evaluator's expected local optimization signal. Evaluator updates should preserve these gaps as optimization signals under policy-induced response set distribution shift.

3.2 Response-Set-Conditioned Evaluation

Preserving this signal requires relative scores that adapt to the candidates being compared. We therefore study how the evaluator input structure affects its capacity to model response-set-specific score gaps. We compare two evaluator function classes. A prompt-only evaluator assigns a fixed score $E(x, y)$ to a prompt-response pair, while a response-set-conditioned evaluator additionally takes the current response set as input and assigns a score $E(x, y \mid \mathcal{C})$. Formally, consider

$$\mathcal{E}_{\mathrm{prompt}} = \{E : (x, y) \mapsto E(x, y) \in \mathbb{R}\}, \qquad \mathcal{E}_{\mathrm{set}} = \{E : (x, y, \mathcal{C}) \mapsto E(x, y \mid \mathcal{C}) \in \mathbb{R}\}. \tag{8}$$

A prompt-only evaluator is the special case satisfying $E(x, y \mid \mathcal{C}) = E(x, y)$. Therefore, $\mathcal{E}_{\mathrm{prompt}} \subseteq \mathcal{E}_{\mathrm{set}}$. To compare the two function classes, we define the local ranking loss functional

$$\mathcal{L}_{x,\mathcal{C}}(E) = \mathbb{E}_{(y^+,y^-)\sim P(\cdot|x,\mathcal{C})}\left[\ell\left(E(x, y^+ \mid \mathcal{C}) - E(x, y^- \mid \mathcal{C})\right)\right], \tag{9}$$

where ℓ is any fixed ranking loss non-increasing in the score gap. The function-class inclusion gives

$$\inf_{E\in\mathcal{E}_{\mathrm{prompt}}} \mathbb{E}_{\mathcal{C}\sim\pi_\theta(\cdot|x)}\left[\mathcal{L}_{x,\mathcal{C}}(E)\right] \geq \inf_{E\in\mathcal{E}_{\mathrm{set}}} \mathbb{E}_{\mathcal{C}\sim\pi_\theta(\cdot|x)}\left[\mathcal{L}_{x,\mathcal{C}}(E)\right]. \tag{10}$$

Equation 10 shows that a response-set-conditioned evaluator has no larger optimal expected local ranking loss than a prompt-only evaluator. By modeling response-set-specific gaps while retaining all prompt-only functions, $\mathcal{E}_{\mathrm{set}}$ provides the basis for preserving relative score gaps.

3.3 Bilevel Policy-Evaluator Updates under Policy-Induced Response-Set Shift

The remaining issue is training-time response-set shift, since policy optimization changes the response sets on which relative gaps must be preserved. As policy optimization improves generations, candidate responses become closer in quality. Reward models and LLM judges are known to be less reliable on such fine-grained comparisons (Tan et al., 2025; Lambert et al., 2024; Li et al., 2024). Consequently, an evaluator trained on earlier response sets may provide weaker relative signals on current policy samples. Therefore, the evaluator must be updated along with policy-induced response sets to maintain the relative scoring structure required for policy optimization.

This joint update requirement naturally yields a coupled optimization problem. The policy update depends on response-level scores produced by the current evaluator. Conversely, the evaluator update depends on response sets sampled from the current policy. For evaluator ϕ and policy θ at round t, we formalize this mutual dependence between the two objectives as the following bilevel problem:

$$\max_{\theta} \quad \mathcal{J}^t_{\mathrm{policy}}(\theta; \phi^\star_t) \qquad \text{s.t.} \quad \phi^\star_t \in \arg\max_{\phi} \mathcal{J}^t_{\mathrm{evaluator}}(\phi; \theta_{t-1}). \tag{11}$$

This coupling makes evaluator learning track policy-induced response-set shift during optimization. It preserves relative discriminability, keeping evaluator scores informative for policy optimization.

3.4 DYNAMICRUBRIC

We instantiate this bilevel formulation with **DynamicRubric**, a co-evolution framework realized through a two-stage optimization loop. At each round, DynamicRubric first fixes the current policy and adapts the evaluator on response sets sampled from that policy. It then fixes the updated evaluator and optimizes the policy using the resulting scores. As illustrated in Figure 1, the evaluator consists of a learnable **DR-Generator** and a frozen **DR-Verifier**. Given a prompt and a response set sampled from **DR-Policy**, DR-Generator produces weighted rubric items that define evaluation criteria and their relative importance. DR-Verifier applies these rubrics to each response and returns binary outcomes for each criterion. The weighted outcomes are aggregated into response-level scores.

Formally, given $(x, \mathcal{C})$, DR-Generator outputs a rubric list $\{(r_m, w_m)\}_{m=1}^{M}$. Given a response y, DR-Verifier returns a yes/no judgment for each rubric item, denoted by $v_m(y; \mathcal{C}) \in \{0, 1\}$. Here, each r_m is a rubric item and $w_m \in \{1, \cdots, 5\}$ is its weight. We do not fix M or prescribe item weights, allowing DR-Generator to determine both during training. Aggregating rubric-level outcomes by normalized weights gives the response-level evaluator used for policy supervision:

$$E_\phi(x, y \mid \mathcal{C}) = \frac{\sum_{m=1}^{M} w_m v_m(y; \mathcal{C})}{\sum_{m=1}^{M} w_m}. \tag{12}$$

At round t, DynamicRubric fixes the current policy θ_{t-1} and samples a response set $\mathcal{C} = \{y_i\}_{i=1}^{K} \sim \pi_{\theta_{t-1}}(\cdot \mid x)$. These online samples typically lack ranking labels. We therefore use a direction-free signal that encourages the generated rubrics to separate responses within the set. For each rubric item r_m, define $\bar{v}_m = \frac{1}{K} \sum_{i=1}^{K} v_m(y_i; \mathcal{C})$ and the discriminability reward over $\mathcal{C}$ as

$$R_{\text{disc}}(x, \mathcal{C}; \phi) = \frac{\sum_{m=1}^{M} w_m \, \bar{v}_m(1 - \bar{v}_m)}{\sum_{m=1}^{M} w_m}. \tag{13}$$

This reward is the weighted Bernoulli variance of rubric outcomes on $\mathcal{C}$. It encourages DR-Generator to produce rubric items that induce non-trivial yes/no splits over the current response set. Averaging this reward over prompts and current-policy response sets gives the **discriminability objective**:

$$\mathcal{J}_{\text{disc}}^{t}(\phi; \theta_{t-1}) = \mathbb{E}_{x \sim \mathcal{D}} \mathbb{E}_{\mathcal{C} \sim \pi_{\theta_{t-1}}(\cdot|x)} \left[R_{\text{disc}}(x, \mathcal{C}; \phi) \right]. \tag{14}$$

Optimizing this objective alone encourages separations, but does not determine whether they are quality-aligned. Indeed, flipping all yes/no outcomes preserves R_{disc} while reversing the induced relative score gaps. Thus, the objective can also reward superficial splits. Updating the evaluator therefore requires a direction-aware ranking signal to calibrate which separations are useful.

Let $\mathcal{A}$ denote ranked anchor data. Each anchor example contains a prompt x and a fully ranked anchor response list $\mathcal{C}_{\text{anchor}} = (y_1^0, \ldots, y_n^0)$ satisfying $y_1^0 \succ y_2^0 \succ \cdots \succ y_n^0$. Given a conditioning response set $\mathcal{C}$, DR-Generator observes only $(x, \mathcal{C})$. The rubrics generated from $\mathcal{C}$ are then applied to $\mathcal{C}_{\text{anchor}}$. The induced score gaps provide preference-alignment supervision, calibrating local separations without exposing anchor-specific shortcuts. For each anchor response $y_i^0 \in \mathcal{C}_{\text{anchor}}$, define $S_i = E_\phi(x, y_i^0 \mid \mathcal{C})$. The anchor reward uses a Bradley–Terry style (Bradley & Terry, 1952) contrastive ranking objective with Rank-Biased Precision style (Moffat & Zobel, 2008) weighting:

$$R_{\text{anchor}}(x, \mathcal{C}, \mathcal{C}_{\text{anchor}}; \phi) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{\log \sigma(S_i - S_j)}{(j - i) \times 2^{i-1}}. \tag{15}$$

Here, $\sigma(\cdot)$ is the logistic sigmoid. The term $\log \sigma(S_i - S_j)$ encourages score gaps in the anchor ranking direction. The weighting scheme increases the weight on top-ranked responses and emphasizes fine-grained distinctions between nearby ranked responses. The corresponding **anchor objective** is

$$\mathcal{J}_{\text{anchor}}^{t}(\phi; \theta_{t-1}) = \mathbb{E}_{(x, \mathcal{C}_{\text{anchor}}) \sim \mathcal{A}} \mathbb{E}_{\mathcal{C} \sim \pi_{\theta_{t-1}}(\cdot|x)} \left[R_{\text{anchor}}(x, \mathcal{C}, \mathcal{C}_{\text{anchor}}; \phi) \right]. \tag{16}$$

Combining these two objectives gives the **evaluator objective**

$$\mathcal{J}_{\text{evaluator}}^{t}(\phi; \theta_{t-1}) = \mathcal{J}_{\text{disc}}^{t}(\phi; \theta_{t-1}) + \lambda \mathcal{J}_{\text{anchor}}^{t}(\phi; \theta_{t-1}), \tag{17}$$

where λ controls the relative scale of the anchor reward. Although R_{anchor} is non-positive, maximizing it directly improves agreement with the anchor ranking. The evaluator objective serves as a

tractable proxy for the **relative score gap** $\Delta_{\text{gap}}(E_\phi; \theta_{t-1})$ **in Equation 7**. Appendix I further shows that there exist constants $c_{\text{disc}}, c_{\text{anchor}} > 0$ and a t-dependent residual ϵ_t such that

$$\Delta_{\text{gap}}(E_\phi; \theta_{t-1}) \geq c_{\text{disc}}\, \mathcal{J}^t_{\text{disc}}(\phi; \theta_{t-1}) + c_{\text{anchor}}\, \mathcal{J}^t_{\text{anchor}}(\phi; \theta_{t-1}) + \epsilon_t. \tag{18}$$

Therefore, maximizing this combination raises the lower bound on the target relative score gap.

After the evaluator update, the **policy objective** uses the response-level reward from the evaluator:

$$\mathcal{J}^t_{\text{policy}}(\theta; \phi_t) = \mathbb{E}_{x\sim\mathcal{D}}\mathbb{E}_{\mathcal{C}\sim\pi_\theta(\cdot|x)}\left[\frac{1}{|\mathcal{C}|}\sum_{y\in\mathcal{C}} E_{\phi_t}(x, y \mid \mathcal{C})\right]. \tag{19}$$

Thus, with θ_0 denoting the initial policy model, the **overall training procedure** is

$$\begin{cases} \phi_t \in \arg\max_{\phi}\ \mathcal{J}^t_{\text{evaluator}}(\phi; \theta_{t-1}), & t \geq 1, \\ \theta_t \in \arg\max_{\theta}\ \mathcal{J}^t_{\text{policy}}(\theta; \phi_t), & t \geq 1, \end{cases} \tag{20}$$

DynamicRubric lets an LLM self-evolve using only ranked anchor data. DR-Generator, DR-Verifier, and DR-Policy can be initialized as separate copies of the same backbone model. The evaluator then learns to supervise the policy without an external reward model or judge.

4 Experiments

4.1 Experimental Setup

Datasets. We use Nectar (Zhu et al., 2023), an open-domain dataset with seven ranked responses per prompt, as anchor data. For evaluator evaluation, we use four pairwise and two listwise preference benchmarks and report five-run averages (AVG@5). Policy evaluation uses four open-ended generation benchmarks. Dataset and benchmark details are provided in Appendices B and C, respectively.

Models. We initialize DR-Policy and DR-Generator from independent Qwen3-8B (Yang et al., 2025) copies and optimize both with GRPO (Shao et al., 2024). For DR-Policy, GRPO turns evaluator scores into relative advantages within each sampled set, aligning policy optimization with the score-gap view. For DR-Generator, discriminability and anchor rewards are combined at the advantage level, which reduces scale mismatch and allows us to set $\lambda = 1$. Training details are provided in Appendix D. We repeat the full pipeline with Llama-3.1-8B-Instruct (Grattafiori et al., 2024) in Appendix F.3, demonstrating DynamicRubric's robustness across different model backbones.

4.2 Evaluator Optimization with DynamicRubric

We first examine whether DynamicRubric improves fine-grained preference evaluation. Our evaluator uses a DR-Generator initialized from Qwen3-8B and trained with DynamicRubric, denoted **DR-Generator-8B**. We compare it with the larger same-family Qwen3-32B used as a dynamic rubric generator, and with strong scalar reward models, including Skywork-Reward-Gemma2-27B (Liu et al., 2024) and Nemotron-70B-Reward-Multilingual (Wang et al., 2025). To control for response-set access, we also evaluate scalar reward models that receive the same response set during scoring. This setup isolates the effect of rubric generator training by controlling for response-set access.

Table 1 shows that DR-Generator-8B consistently improves over zero-shot dynamic rubric evaluators, including the larger Qwen3-32B generator, on all accuracy metrics. In contrast, adding response-set access improves three of eight metrics for Skywork-27B and one for Nemotron-70B. Most other scores decline, so response-set access alone does not account for DynamicRubric's gains. DR-Generator-8B outperforms the 27B reward model and remains competitive with the 70B reward model. We also include qualitative analysis and a case study in Appendices G and H to examine whether DynamicRubric training turns response-set access into high-quality rubrics that identify decisive differences. The analysis shows that training yields more discriminative criteria, and the case study illustrates how these criteria produce clear score gaps and the correct top-ranked response.

Table 1: Evaluator performance on preference benchmarks. "w/ responses" marks scalar reward models given the response set. **DR-Generator-8B** uses a Qwen3-8B backbone. Acc. is strict accuracy: top-score ties count as errors. nDCG is for listwise benchmarks. **Bold** marks the overall best and underlining the best dynamic rubric generator with the same verifier (Appendix F.1).

Evaluator	Size	JudgeBench	Personalized-RB	RM-Bench	UltraFeedback	AEOLLM		LMSYS-Chat-1M	
		Acc.↑	Acc.↑	Acc.↑	Acc.↑	Acc.↑	nDCG↑	Acc.↑	nDCG↑
Scalar Reward Models									
Skywork-27B	27B	52.1	64.7	67.7	63.4	43.8	0.826	21.0	0.720
Skywork-27B w/ responses	27B	56.1	60.6	65.8	65.2	37.5	0.805	21.6	0.717
Nemotron-70B	70B	**66.0**	68.6	**85.2**	72.8	**57.5**	0.868	20.4	0.718
Nemotron-70B w/ responses	70B	64.8	61.9	83.2	**77.4**	**57.5**	0.867	17.2	0.706
Dynamic Rubric Methods									
Qwen3-8B zero-shot	8B	36.4	53.7	52.2	54.7	17.8	0.894	12.0	0.762
Qwen3-32B zero-shot	32B	46.4	67.6	64.7	64.6	29.7	**0.902**	15.1	0.763
DR-Generator-8B	8B	57.4	**73.3**	69.8	68.7	51.2	0.898	**28.0**	**0.779**

Table 2: Policy performance on open-domain generation benchmarks across different supervision methods. AlpacaEval2 and ArenaHardv2.0 report win rates, while WildBench and WritingBench report scores on their standard scales. Higher scores are better, and the best in each column is in **bold**.

Supervision Method	Supervisor Size	Backbone	AlpacaEval2	Arena Hardv2.0	WildBench	Writing Bench
Baselines						
Native	–	Qwen3-8B	38.5	9.0	16.4	57.8
Native	–	Qwen3-32B	50.2	20.8	29.7	62.0
Scalar Reward Models						
Skywork-27B	27B	Qwen3-8B	50.6	15.3	22.3	60.2
Skywork-27B w/ responses	27B	Qwen3-8B	48.2	12.3	20.5	58.6
Nemotron-70B	70B	Qwen3-8B	35.2	10.3	17.5	56.2
Nemotron-70B w/ responses	70B	Qwen3-8B	41.9	9.2	18.9	58.2
Static Rubric						
RaR (Gunjal et al., 2025)	235B	Qwen3-8B	58.5	15.7	25.2	61.6
Dynamic Rubric						
DR-Generator-8B	8B	Qwen3-8B	**67.1**	**21.0**	**30.7**	**64.3**

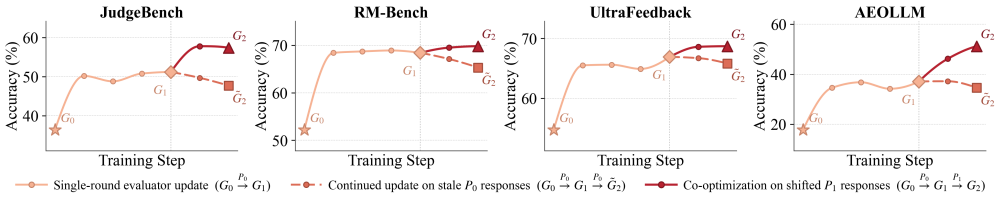
4.3 Policy Optimization with DynamicRubric

Next, we evaluate whether these finer score gaps translate into effective supervision for downstream DR-Policy optimization. We compare against other supervision approaches including scalar reward models, their response-set-conditioned variants, and Rubric-as-Reward (Gunjal et al., 2025), a static rubric approach that uses Qwen3-235B-A22B-Instruct-2507 to generate rubrics from the prompt alone. Qwen3-32B is also included as a strong same-family policy baseline.

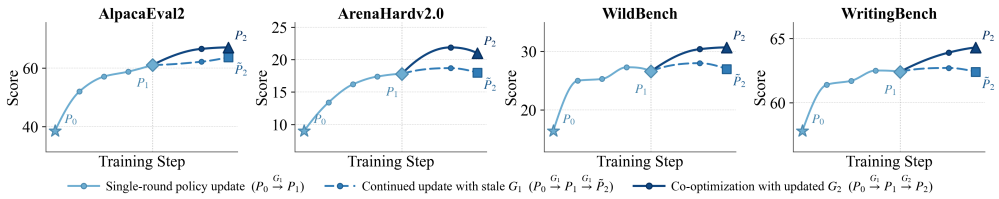
Table 2 shows that DR-Generator-8B supervision produces a DR-Policy that consistently outperforms policies trained with larger reward models, static rubric supervision from a 235B generator, and the strong 32B baseline across all benchmarks. We use DeepSeek-V4-Flash (DeepSeek-AI, 2026) as the judge. Appendix F.2 re-evaluates policies with GPT-4.1 and reports consistent gains, supporting the reliability of the evaluation. Together, these results show that DynamicRubric converts response-set-conditioned evaluator gains into effective reward signals for policy optimization.

4.4 Continuous Co-Evolution of Evaluator and Policy

We test whether continuous co-evolution improves evaluator adaptation and policy optimization through two controlled comparisons. We initialize G_0 and P_0 as separate copies of the native Qwen3-8B backbone. The first round updates G_0 to G_1 on P_0 responses and uses G_1 to optimize P_0 into



(a) DR-Generator optimization on original versus updated DR-Policy response distribution.



(b) DR-Policy optimization with dynamic rubric supervision from previous versus updated DR-Generators.

Figure 2: Effect of evaluator–policy co-evolution in DynamicRubric with the same model backbone.

P_1, shifting the policy-induced response distribution. For evaluator adaptation, we train G_1 on either stale P_0 responses, $G_1 \xrightarrow{P_0} \widetilde{G}_2$, or current P_1 responses, $G_1 \xrightarrow{P_1} G_2$. For policy optimization, we train P_1 with either stale G_1 supervision, $P_1 \xrightarrow{G_1} \widetilde{P}_2$, or updated G_2 supervision, $P_1 \xrightarrow{G_2} P_2$.

Figure 2 shows that adapting DR-Generator to current P_1 responses consistently improves evaluator performance across all benchmarks. Continued training on stale P_0 responses instead plateaus or degrades. Updated G_2 also produces stronger gains than stale G_1 across all generation benchmarks. The two controlled comparisons establish the core design principle of DynamicRubric: evaluator learning must track the evolving policy-induced response distribution, and policy optimization must use an evaluator adapted to that distribution. Tracking this shift preserves discriminative score gaps and prevents stale supervision from limiting policy improvement in subsequent rounds.

4.5 Deployment in Production

Finally, we fully deploy a model optimized by DynamicRubric for AI answering in an online production environment. It replaces the previous production model and serves all online traffic, handling tens of millions of requests per day. Online A/B tests against the previous model show statistically significant relative improvements in total search volume, user duration, and absolute positive user behaviors. These gains demonstrate that DynamicRubric scales to a much larger production model. In production, DynamicRubric also enables continual improvement by refreshing the anchors. These updates recalibrate evaluator feedback for subsequent policy optimization. The evaluator and policy thus co-evolve in line with real user preferences.

5 Analysis

5.1 Ablation Study on Evaluator Optimization Objectives

We ablate two objective terms used to optimize the evaluator in DynamicRubric: the discriminability objective $\mathcal{J}_{\text{disc}}$ and the anchor objective $\mathcal{J}_{\text{anchor}}$. We compare the full objective with variants that omit either term. All variants start from the same DR-Generator and are optimized on response sets sampled from the same policy for the same number of steps, isolating the effect of each objective.

Table 3 shows that the full objective achieves the highest macro average across benchmarks. Removing $\mathcal{J}_{\text{anchor}}$ leaves the optimization signal vulnerable to score gaps that do not reflect response quality. Removing $\mathcal{J}_{\text{disc}}$ weakens evaluator adaptation to response sets sampled from the current policy. This pattern indicates that separation and ranking calibration provide complementary supervision for DR-Generator, and their combination yields a stronger evaluator across benchmarks.

Table 3: Ablation of evaluator optimization objectives. In each column, the best is in **bold**.

Evaluator Objective	$\mathcal{J}_{\text{disc}}$	$\mathcal{J}_{\text{anchor}}$	JudgeBench Acc.↑	Personalized-RB Acc.↑	RM-Bench Acc.↑	Ultra Feedback Acc.↑	AEOLLM Acc.↑	LMSYS-Chat-1M Acc.↑	Average Macro↑
w/o anchor objective	✓	-	56.1	72.4	**69.8**	67.9	47.3	27.7	56.9
w/o discriminability objective	-	✓	50.0	**73.8**	66.3	66.8	38.8	20.7	52.7
Full objective	✓	✓	**57.4**	73.3	**69.8**	**68.7**	**51.2**	**28.0**	**58.1**

Table 4: Computation overhead. GPU allocation and GPU hours for policy optimization are relative to the rule-based baseline. RaR includes offline rubric generation.

Supervision Method	Supervisor Size	Verifier	Minimal GPU Allocation	Actual GPU Hours
Rule-based	0B	No	1.0×	1.0×
Nemotron-70B w/ responses	70B	No	1.5×	3.5×
RaR (Gunjal et al., 2025)	235B	Yes	max(1.0×, 235B rubric generator GPU allocation)	2.5× + 235B rubric generator cost
DR-Generator-8B	8B	Yes	1.0×	3.0×

5.2 DynamicRubric Policy Gains Transfer to Verifiable Tasks

DR-Policies are optimized for open-ended generation. Prior work reports that open-ended generation optimization can cause regressions on reasoning and coding tasks (Ouyang et al., 2022; Kirk et al., 2024). This raises the question of whether the benefits of DynamicRubric extend to reasoning abilities. We evaluate DR-Policies and report AVG@5 on mathematics, scientific QA, and coding benchmarks detailed in Appendix C. Figure 3 shows that DR-Policies consistently improve over the initial policy across these out-of-distribution tasks, indicating that DynamicRubric improves open-ended generation and that its policy-level benefits transfer to verifiable reasoning and coding tasks.

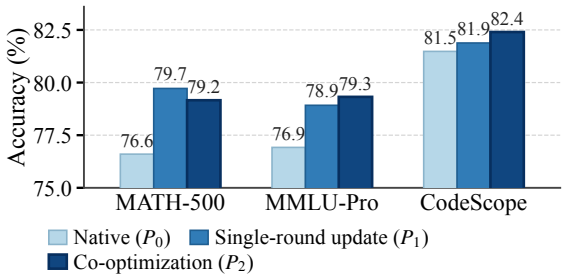


Figure 3: Transfer of DynamicRubric policy gains from open-ended generation to verifiable tasks.

5.3 Computational Overhead

DynamicRubric realizes self-evolution without introducing additional GPU allocation. Because the generator, verifier, and policy are separate copies of one backbone, SGLang runs all three on the same nodes, reloading their weights between pipeline stages. Table 4 shows that DynamicRubric's minimum GPU allocation equals the cheapest baseline's and stays below the rest, while its total GPU hours also remain competitive. Appendix E gives the full configurations of all stages.

6 Conclusion

We identify a central bottleneck in evaluator-guided policy optimization: as current-policy responses become close in quality, effective updates require evaluators to maintain discriminability. We formalize this requirement through a probability allocation view, where moving probability mass between two responses yields a gain equal to their evaluator score gap. This yields a simple principle: evaluators should evolve with the policies they supervise, adapting to current-policy response sets before the next policy update. We propose **DynamicRubric** to instantiate this principle with response-set-conditioned rubrics and realize the self-evolution of the LLM. Experiments show gains in evaluation, open-ended generation, and verifiable tasks, highlighting relative discriminability on the policy-induced response distribution as central to effective post-training evaluators.

AI USE STATEMENT

We used AI tools for language polishing and GitHub Copilot for coding assistance. The authors manually reviewed all content and take full responsibility for the final manuscript and code.

ETHICS STATEMENT

Our benchmark experiments use existing datasets and pretrained models. The production experiments use a proprietary language model whose identity and technical details remain commercially confidential.

REPRODUCIBILITY STATEMENT

The method and training algorithm are described in Section 3 and Appendix A. Appendices B, C, D, and E document data processing, evaluation protocols, training settings, and computational overhead, respectively. Appendix J provides the prompt templates. Our code is included with the submission.

REFERENCES

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms, 2024. URL https://arxiv.org/abs/2402.14740.

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, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Kamile Lukosuite, Liane Lovitt, Michael Sellitto, Nelson Elhage, Nicholas Schiefer, Noemi Mercado, Nova DasSarma, Robert Lasenby, Robin Larson, Sam Ringer, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Tamera Lanham, Timothy Telleen-Lawton, Tom Conerly, Tom Henighan, Tristan Hume, Samuel R. Bowman, Zac Hatfield-Dodds, Ben Mann, Dario Amodei, Nicholas Joseph, Sam McCandlish, Tom Brown, and Jared Kaplan. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.

Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

Junjie Chen, Haitao Li, Zhumin Chu, Yiqun Liu, and Qingyao Ai. Overview of the ntcir-18 automatic evaluation of llms (aeollm) task, 2025. URL https://arxiv.org/abs/2503.13038.

Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Wei Zhu, Yuan Ni, Guotong Xie, Zhiyuan Liu, and Maosong Sun. Ultrafeedback: Boosting language models with high-quality feedback. 2023.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026.

Hongxin Ding, Baixiang Huang, Yue Fang, Weibin Liao, Zheng Li, Jinyang Zhang, Zhijing Wu, Junfeng Zhao, and Yasha Wang. EvoRubrics: Dynamic rubrics as rewards via adversarial co-evolution for LLM reinforcement learning. *arXiv preprint arXiv:2606.23038*, 2026. URL https://arxiv.org/abs/2606.23038.

Yann Dubois, Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Alpacafarm: A simulation framework for methods that learn from human feedback, 2023.

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

Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. In *International Conference on Machine Learning*, pp. 10835–10866. PMLR, 2023.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James

Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

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

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

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, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu

Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025a. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

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, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025b. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002.

Seungone Kim, Jamin Shin, Yejin Cho, Joel Jang, Shayne Longpre, Hwaran Lee, Sangdoo Yun, Seongjin Shin, Sungdong Kim, James Thorne, and Minjoon Seo. Prometheus: Inducing fine-grained evaluation capability in language models, 2024. URL https://arxiv.org/abs/2310.08491.

Robert Kirk, Ishita Mediratta, Christoforos Nalmpantis, Jelena Luketina, Eric Hambro, Edward Grefenstette, and Roberta Raileanu. Understanding the effects of rlhf on llm generalisation and diversity, 2024. URL https://arxiv.org/abs/2310.06452.

Hynek Kydlíček. Math-Verify: Math Verification Library. URL https://github.com/huggingface/math-verify.

Nathan Lambert, Valentina Pyatkin, Jacob Morrison, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, Noah A. Smith, and Hannaneh Hajishirzi. Rewardbench: Evaluating reward models for language modeling, 2024. URL https://arxiv.org/abs/2403.13787.

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

Xuechen Li, Tianyi Zhang, Yann Dubois, Rohan Taori, Ishaan Gulrajani, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Alpacaeval: An automatic evaluator of instruction-following models. `https://github.com/tatsu-lab/alpaca_eval`, 5 2023.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

Chris Yuhao Liu, Liang Zeng, Jiacai Liu, Rui Yan, Jujie He, Chaojie Wang, Shuicheng Yan, Yang Liu, and Yahui Zhou. Skywork-reward: Bag of tricks for reward modeling in llms. *arXiv preprint arXiv:2410.18451*, 2024.

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

Yantao Liu, Zijun Yao, Rui Min, Yixin Cao, Lei Hou, and Juanzi Li. Rm-bench: Benchmarking reward models of language models with subtlety and style. In *International Conference on Learning Representations*, volume 2025, pp. 44323–44355, 2025.

Qiyao Ma, Dechen Gao, Rui Cai, Boqi Zhao, Hanchu Zhou, Junshan Zhang, and Zhe Zhao. Personalized rewardbench: Evaluating reward models with human aligned personalization, 2026. URL `https://arxiv.org/abs/2604.07343`.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016. URL `https://arxiv.org/abs/1602.01783`.

Alistair Moffat and Justin Zobel. Rank-biased precision for measurement of retrieval effectiveness. *ACM Transactions on Information Systems (TOIS)*, 27(1):1–27, 2008.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022. URL `https://arxiv.org/abs/2203.02155`.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL `https://arxiv.org/abs/1707.06347`.

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

Hwanjun Song, Hang Su, Igor Shalyminov, Jason Cai, and Saab Mansour. Finesure: Fine-grained summarization evaluation using llms. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 906–922, 2024.

Sijun Tan, Siyuan Zhuang, Kyle Montgomery, William Tang, Alejandro Cuadron, Chenguang Wang, Raluca Popa, and Ion Stoica. Judgebench: A benchmark for evaluating llm-based judges. In *International Conference on Learning Representations*, volume 2025, pp. 63277–63303, 2025.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark, 2024. URL `https://arxiv.org/abs/2406.01574`.

Zhilin Wang, Jiaqi Zeng, Olivier Delalleau, Hoo-Chang Shin, Felipe Soares, Alexander Bukharin, Ellie Evans, Yi Dong, and Oleksii Kuchaiev. HelpSteer3-Preference: Open human-annotated preference data across diverse tasks and languages, 2025. URL https://arxiv.org/abs/2505.11475.

Junyu Wu, Weiming Chang, Xiaotao Liu, Guanyou He, Tingfeng Xian, Haoqiang Hong, Boqi Chen, Hongtao Tian, Tao Yang, Yunsheng Shi, Feng Lin, Ting Yao, and Jiatao Xu. WeChat-YATT: A Scalable, Simple, Efficient, and Production Ready Training Library, August 2025a.

Yuning Wu, Jiahao Mei, Ming Yan, Chenliang Li, Shaopeng Lai, Yuran Ren, Zijia Wang, Ji Zhang, Mengyue Wu, Qin Jin, and Fei Huang. Writingbench: A comprehensive benchmark for generative writing, 2025b. URL https://arxiv.org/abs/2503.05244.

Weixiang Yan, Haitian Liu, Yunkun Wang, Yunzhe Li, Qian Chen, Wen Wang, Tingyu Lin, Weishan Zhao, Li Zhu, Hari Sundaram, and Shuiguang Deng. Codescope: An execution-based multilingual multitask multidimensional benchmark for evaluating llms on code understanding and generation, 2024. URL https://arxiv.org/abs/2311.08588.

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, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Seonghyeon Ye, Doyoung Kim, Sungdong Kim, Hyeonbin Hwang, Seungone Kim, Yongrae Jo, James Thorne, Juho Kim, and Minjoon Seo. Flask: Fine-grained language model evaluation based on alignment skill sets. *arXiv preprint arXiv:2307.10928*, 2023.

Bill Yuchen Lin, Yuntian Deng, Khyathi Chandu, Faeze Brahman, Abhilasha Ravichander, Valentina Pyatkin, Nouha Dziri, Ronan Le Bras, and Yejin Choi. Wildbench: Benchmarking llms with challenging tasks from real users in the wild. *arXiv e-prints*, pp. arXiv–2406, 2024.

Junkai Zhang, Zihao Wang, Lin Gui, Swarnashree Mysore Sathyendra, Jaehwan Jeong, Victor Veitch, Wei Wang, Yunzhong He, Bing Liu, and Lifeng Jin. Chasing the tail: Effective rubric-based reward modeling for large language model post-training. *arXiv preprint arXiv:2509.21500*, 2025.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023. URL https://arxiv.org/abs/2306.05685.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Tianle Li, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zhuohan Li, Zi Lin, Eric P. Xing, Joseph E. Gonzalez, Ion Stoica, and Hao Zhang. Lmsys-chat-1m: A large-scale real-world llm conversation dataset. In *International Conference on Learning Representations*, volume 2024, pp. 22225–22257, 2024.

Banghua Zhu, Evan Frick, Tianhao Wu, Hanlin Zhu, and Jiantao Jiao. Starling-7b: Improving llm helpfulness harmlessness with rlaif, November 2023.

Algorithm 1 DynamicRubric Training

Require: Initial policy $P_0 = \pi_{\theta_0}$; initial generator $G_0 = G_{\phi_0}$; frozen verifier V; prompt distribution $\mathcal{D}$; ranked anchor data $\mathcal{A}$; rounds T; response-set size K; anchor weight λ.
Ensure: Trained generator G_T and trained policy P_T.

for $t = 1, \ldots, T$ **do**
Evaluator update: $G_{t-1} \to G_t$ **using samples from** P_{t-1}**.**
  Initialize $\phi \leftarrow \phi_{t-1}$.
  for generator update steps **do**
    Sample $(x, \mathcal{C}_{\text{anchor}}) \sim \mathcal{A}$ and current-policy responses $\mathcal{C} = \{y_i\}_{i=1}^{K}$ with $y_i \sim P_{t-1}(\cdot \mid x)$.
    Generate rubrics $\mathcal{R}_\phi(x, \mathcal{C}) = \{(r_m, w_m)\}_{m=1}^{M}$.
    Use V to verify responses in $\mathcal{C}$ and compute $R_{\text{disc}}(x, \mathcal{C}; \phi)$.
    Use the same rubrics to verify $\mathcal{C}_{\text{anchor}}$ and compute $R_{\text{anchor}}(x, \mathcal{C}, \mathcal{C}_{\text{anchor}}; \phi)$.
    Update ϕ to maximize $R_{\text{disc}} + \lambda R_{\text{anchor}}$.
  end for
  Set $\phi_t \leftarrow \phi$ and $G_t \leftarrow G_{\phi_t}$.
Policy update: $P_{t-1} \to P_t$ **using the fixed generator** G_t**.**
  Initialize $\theta \leftarrow \theta_{t-1}$.
  for policy update steps **do**
    Sample $x \sim \mathcal{D}$ and responses $\mathcal{C} = \{y_i\}_{i=1}^{K}$ with $y_i \sim \pi_\theta(\cdot \mid x)$.
    Generate rubrics $\mathcal{R}_{\phi_t}(x, \mathcal{C})$ using fixed G_t.
    Use V to compute scores $s_i = E_{\phi_t}(x, y_i \mid \mathcal{C})$.
    Update θ with the policy optimizer using the rubric-based scores.
  end for
  Set $\theta_t \leftarrow \theta$ and $P_t \leftarrow \pi_{\theta_t}$.
end for
return G_T and P_T.

A DynamicRubric Training Algorithm

Algorithm 1 summarizes DynamicRubric training. The first sampled response sets come from the initial policy P_0, so the first update adapts G_0 to G_1. The updated generator G_1 then supervises the first policy update from P_0 to P_1.

For GRPO, rubric scores are converted into group-normalized advantages, so the policy update depends on within-set relative scores rather than absolute reward scale.

B Dataset Details

B.1 Dataset Details for Training DR-Generator

For DR-Generator optimization, each training instance contains a user prompt and a candidate response set. We apply lightweight filtering to ensure that the generator observes sufficiently informative inputs while avoiding response sets dominated by degenerate model outputs.

We first apply length-based filtering. Each example must contain at least seven candidate responses. The user prompt and the top-ranked response must each contain more than 50 characters. We then apply response-set-level filtering to the first seven responses. A response set is discarded if it contains any repetitive generation, if at least four responses are degenerate, if fewer than four responses are useful, or if at least four responses consist mostly of boilerplate. Here, degenerate responses include broken text, plain refusals, greeting-only replies, and clarification-only replies, while useful responses include substantive answers, refusals with helpful alternatives, and other nontrivial responses. After filtering, we obtain 24K training samples for DR-Generator optimization.

B.2 Dataset Details for Training DR-Policy

For DR-Policy optimization, each training instance is a user prompt. We apply lightweight filtering to retain prompts that are sufficiently informative and whose associated source responses are not degenerate. Each example must contain at least seven associated responses. For most sources, the user prompt and the top-ranked associated response must each contain more than 50 characters. For short-prompt sources whose prompts are already filtered upstream, we exempt the prompt from the length constraint, while still requiring the top-ranked response to be longer than 50 characters.

We further discard examples whose top-ranked associated response contains broken text, is repetitive, is a plain refusal, or consists only of a greeting or clarification request. This filtering ensures that DR-Policy is optimized on prompts for which meaningful response generation can be supervised by the current DR-Generator.

After filtering, we retain 36K prompts for DR-Policy optimization.

C Benchmark Details

We evaluate DynamicRubric along three axes. First, evaluator evaluation measures whether the trained DR-Generator produces rubrics that induce accurate response rankings. Second, open-ended generation evaluation measures whether DR-Policy improves general instruction-following and conversational quality. Third, verifiable task evaluation measures whether optimization with open-ended dynamic rubric supervision preserves capabilities on reasoning-heavy out-of-distribution tasks.

C.1 Benchmark Details for Evaluator Evaluation

For evaluator evaluation, we use pairwise preference benchmarks including JudgeBench (Tan et al., 2025), Personalized-RewardBench (Ma et al., 2026), RM-Bench (Liu et al., 2025), and UltraFeedback (Cui et al., 2023), and listwise preference benchmarks including AEOLLM (Chen et al., 2025) and LMSYS-Chat-1M (Zheng et al., 2024). For pairwise benchmarks, the evaluator is asked to select the preferred response from a candidate pair. For listwise benchmarks, the evaluator assigns scores to multiple candidate responses, and the induced ranking is compared with the annotated ranking.

When evaluating dynamic rubric generators in Section 4.2, we use Qwen3-30B-A3B as the default verifier for all dynamic rubric methods. This choice keeps the verifier fixed while avoiding same-model coupling with the generator baselines, which include Qwen3-8B and Qwen3-32B. Using a separate model as the verifier ensures that no compared generator is evaluated by a verifier initialized from the same model instance or scale. Section F.1 further evaluates the same generators under multiple verifier choices, including Llama-3.1-8B-Instruct and Qwen3 verifiers of different scales. DR-Generator-8B consistently improves over the zero-shot dynamic rubric generators across these verifier choices, including the cross-family Llama verifier, indicating that its gains are not tied to a particular verifier model.

JudgeBench. JudgeBench is a preference benchmark for evaluating LLM-as-a-judge and reward model behavior. We use it as a pairwise evaluator benchmark, where each example contains a prompt, two candidate responses, and an annotated preference.

Personalized-RewardBench. Personalized-RewardBench evaluates reward models under personalized scenarios. We use it as a pairwise evaluator benchmark, where each example contains a prompt, two candidate responses, and an annotated preference.

RM-Bench. RM-Bench is a reward model evaluation benchmark built around response preference comparisons. We use it as a pairwise evaluator benchmark, where each example contains a prompt, two candidate responses, and an annotated preference.

UltraFeedback. UltraFeedback contains prompts paired with multiple model responses and preference-style annotations. We use it as a pairwise evaluator benchmark, where each example contains a prompt, two candidate responses, and an annotated preference.

AEOLLM. AEOLLM is a listwise evaluation benchmark for automatic LLM evaluation. We use it to measure whether the evaluator can rank multiple candidate responses consistently with the annotated ordering. We report both strict top-1 accuracy and nDCG (Järvelin & Kekäläinen, 2002).

LMSYS-Chat-1M. LMSYS-Chat-1M contains real-world user conversations from large-scale chat deployments. Nectar provides the responses and ranking annotations for this subset. We hold out the complete subset for listwise evaluation and draw training data exclusively from the other Nectar subsets to avoid leakage. We report both strict top-1 accuracy and nDCG.

C.2 Benchmark Details for Open-Ended Generation Tasks Evaluation

For policy evaluation, we use DeepSeek-V4-Flash (DeepSeek-AI, 2026) as the judge on four open-domain generation benchmarks: AlpacaEval2 (Li et al., 2023; Dubois et al., 2023; 2024), Arena-Hardv2.0 (Li et al., 2024), WildBench (Yuchen Lin et al., 2024), and WritingBench (Wu et al., 2025b). AlpacaEval2 and ArenaHardv2.0 report win rates, while WildBench and WritingBench report benchmark scores in their standard evaluation scales. All metrics are reported such that higher is better. To assess robustness to the choice of automatic judge, Appendix F.2 further re-evaluates the same policy checkpoints with GPT-4.1 and reports consistent gains.

AlpacaEval2. AlpacaEval2 evaluates instruction-following ability through model-based pairwise comparisons on open-ended user instructions. We use it to measure whether DR-Policy improves general assistant-style response quality.

ArenaHardv2.0. ArenaHardv2.0 contains challenging open-ended prompts designed to stress-test instruction-following models. We use it to evaluate policy performance on difficult conversational and reasoning-heavy user requests.

WildBench. WildBench evaluates models on real-world conversational tasks. We use it to measure whether DR-Policy improves response quality under diverse user-facing interaction scenarios.

WritingBench. WritingBench evaluates long-form writing ability. We use it to test whether DynamicRubric optimization improves policy behavior on tasks requiring coherent, structured, and high-quality written responses.

C.3 Benchmark Details for Verifiable Tasks Evaluation

For verifiable-task evaluation, we use MATH-500 (Lightman et al., 2023) for mathematical reasoning, MMLU-Pro (Wang et al., 2024) for scientific question answering, and CodeScope (Yan et al., 2024) for coding. MATH-500 is evaluated with Math-Verify (Kydlíček), while MMLU-Pro and CodeScope are evaluated by exact match.

MATH-500. MATH-500 is a mathematical reasoning benchmark consisting of competition-style math problems. We use Math-Verify (Kydlíček) (it is Apache-2.0 licensed and we follow its instructions in the repository) to check whether the model's final answer matches the reference answer.

MMLU-Pro. MMLU-Pro is a challenging question-answering benchmark. We use its scientific question-answering setting and evaluate predictions by exact match.

CodeScope. CodeScope is a coding benchmark for evaluating programming ability. We evaluate model outputs by exact match.

D Training Details

We train our models with GRPO (Guo et al., 2025b; Shao et al., 2024) as implemented in WeChat-YATT (Wu et al., 2025a). We use a rollout size of 8 samples per instance, corresponding to $K = 8$ in Equation 1, a global batch size of 256, $\beta = 0.01$, and learning rates of 5×10^{-6} for DR-Generator and 1×10^{-6} for DR-Policy. The maximum input/output lengths are 8K/8K for DR-Policy and

Table 5: Computational overhead for policy optimization under different supervision methods. GPU allocation and GPU hours are relative to the rule-based baseline. Minimal GPU Allocation reports the minimum GPU resources that must be reserved for the training pipeline, and Actual GPU Hours reports the total end-to-end training cost in GPU hours. For Actual GPU Hours, all pipelines are run with an additional $1\times$ auxiliary GPU allocation for acceleration. For DynamicRubric, DR-Generator, DR-Verifier, and DR-Policy are separate copies of one backbone, so SGLang runs all three on the same nodes without additional GPU allocation.

Supervision Method	Supervisor Size	Backbone	Verifier Needed	Minimal GPU Allocation	Actual GPU Hours
Baseline					
Rule-based	0B	Qwen3-8B	No	$1\times$	$1\times$
Scalar Reward Models					
Skywork-27B	27B	Qwen3-8B	No	$1.125\times$	$2.2\times$
Skywork-27B w/ responses	27B	Qwen3-8B	No	$1.125\times$	$2.6\times$
Nemotron-70B	70B	Qwen3-8B	No	$1.5\times$	$2.4\times$
Nemotron-70B w/ responses	70B	Qwen3-8B	No	$1.5\times$	$3.5\times$
Static Rubric					
RaR (Gunjal et al., 2025)	235B	Qwen3-8B	Yes	$\max(1.0\times$, 235B rubric generator GPU allocation)	$2.5\times$ + 235B rubric generator cost
Dynamic Rubric					
DR-Generator-8B	8B	Qwen3-8B	Yes	$1\times$	$3.0\times$

24K/8K for DR-Generator, as DR-Generator scores the full response set from each rollout. We do not constrain the number of generated rubrics M or their weights, but let DR-Generator learn this allocation from data; empirically, it learns reasonable schemes as the experimental results show.

GRPO replaces the separate value function (Schulman et al., 2017; Mnih et al., 2016) with a group-based relative advantage estimation (Ahmadian et al., 2024). For each question q, the policy $\pi_{\theta_{\text{old}}}$ generates G candidate outputs $\{o_i\}_{i=1}^{G}$. The advantage for each token $o_{i,t}$ is computed as

$$\hat{A}_{i,t} = \frac{R_i - \text{mean}(\{R_j\}_{j=1}^{G})}{\text{std}(\{R_j\}_{j=1}^{G})}, \tag{21}$$

where R_i denotes the scalar reward assigned to output o_i. This formulation normalizes rewards within the group. In the native GRPO implementation, the reward is binary and determined by an automatic rule-based verifier:

$$R_i = \begin{cases} 1, & \text{if the verifier returns } \texttt{true} \text{ for output } o_i, \\ 0, & \text{otherwise.} \end{cases} \tag{22}$$

The GRPO objective is defined as

$$\begin{aligned} \mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{q,\{o_i\}} \frac{1}{G} \sum_{i=1}^{G} \frac{1}{|o_i|} \sum_{t=1}^{|o_i|} \Bigg\{ \\ \min\left[r_{i,t}(\theta)\hat{A}_{i,t},\ \text{clip}(r_{i,t}(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_{i,t} \right] \\ - \beta\, \mathbb{D}_{\text{KL}}\left[\pi_\theta \,\|\, \pi_{\text{ref}}\right] \Bigg\}, \end{aligned} \tag{23}$$

where $r_{i,t}(\theta) = \frac{\pi_\theta(o_{i,t}|q,o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t}|q,o_{i,<t})}$ is the token-level probability ratio and β controls the KL penalty strength with respect to a reference policy π_{ref}.

In Section 4.1, we combine two objectives at the advantage level. Formally, that is

$$A_{i,t} = \hat{A}_{i,t}^{disc} + \hat{A}_{i,t}^{anchor}, \tag{24}$$

Table 6: Evaluator performance when dynamic rubric generators are paired with different verifiers. Each block fixes the verifier used to apply generated rubrics and compares Qwen3-8B zero-shot, Qwen3-32B zero-shot, and the trained DR-Generator-8B. Acc. denotes strict top-1 accuracy with ties excluded. Within each verifier block, the best result is in **bold**.

Rubric Generator	Size	JudgeBench Acc.↑	Personalized-RB Acc.↑	RM-Bench Acc.↑	UltraFeedback Acc.↑	AEOLLM Acc.↑	LMSYS-Chat-1M Acc.↑
Verifier: Llama-3.1-8B-Instruct							
Qwen3-8B zero-shot	8B	27.7	38.7	40.3	44.1	21.7	11.9
Qwen3-32B zero-shot	32B	38.4	60.9	58.6	60.0	34.8	16.5
DR-Generator-8B	8B	**50.3**	**68.6**	**63.1**	**63.0**	**43.7**	**26.1**
Verifier: Qwen3-8B							
Qwen3-8B zero-shot	8B	33.1	52.5	49.9	51.8	18.2	11.0
Qwen3-32B zero-shot	32B	42.8	67.8	62.8	62.6	25.3	14.1
DR-Generator-8B	8B	**54.9**	**73.0**	**67.0**	**66.8**	**46.5**	**27.4**
Verifier: Qwen3-30B-A3B							
Qwen3-8B zero-shot	8B	36.4	53.7	52.2	54.7	17.8	12.0
Qwen3-32B zero-shot	32B	46.4	67.6	64.7	64.6	29.7	15.1
DR-Generator-8B	8B	**57.4**	**73.3**	**69.8**	**68.7**	**51.2**	**28.0**
Verifier: Qwen3-32B							
Qwen3-8B zero-shot	8B	35.8	54.6	50.7	53.3	21.3	12.7
Qwen3-32B zero-shot	32B	46.7	69.5	65.4	65.5	32.2	16.5
DR-Generator-8B	8B	**57.2**	**72.7**	**68.0**	**68.2**	**52.5**	**29.1**
Verifier: Qwen3-235B-A22B							
Qwen3-8B zero-shot	8B	40.4	55.0	56.8	55.5	23.1	13.8
Qwen3-32B zero-shot	32B	50.2	69.2	70.5	67.1	34.7	18.1
DR-Generator-8B	8B	**61.0**	**75.5**	**71.6**	**69.9**	**50.7**	**30.1**

where $\hat{A}^*_{i,t}$ is computed as

$$\hat{A}^*_{i,t} = \frac{R^*_i - \text{mean}(\{R^*_j\}_{j=1}^G)}{\text{std}(\{R^*_j\}_{j=1}^G)}, * \in \{anchor, disc\} \tag{25}$$

During training, we use the structured output mode of SGLang to constrain DR-Generator and DR-Verifier to produce the desired JSON format, simplifying extraction. Stage transitions are performed within an epoch.

E Computational Overhead

Table 5 gives the full configurations of all stages. All numbers are relative to the rule-based baseline using the same Qwen3-8B policy backbone. Minimal GPU Allocation denotes the minimum GPU resources that must be reserved to run the corresponding training pipeline. Actual GPU Hours denotes the total end-to-end GPU-hour cost under a common accelerated setting, where every pipeline receives an additional $1\times$ auxiliary GPU allocation for inference or scoring acceleration.

Scalar reward models do not require a separate verifier, but larger reward models increase the minimum allocated GPU resources. Conditioning scalar reward models on the full response set further increases total GPU hours because the scoring input becomes longer. Static rubric supervision requires a verifier to apply generated rubrics, while its reported cost includes the additional offline cost of generating rubrics for all training prompts with a 235B model.

DynamicRubric also requires rubric verification, but DR-Generator, DR-Verifier, and DR-Policy are separate copies of one backbone, so SGLang runs all three on the same nodes, reloading their weights between pipeline stages. DynamicRubric thus matches the rule-based baseline in minimal GPU allocation, while its additional total GPU hours come from online rubric generation and verification.

Table 7: Policy performance on open-domain generation benchmarks across different supervision methods, evaluated with GPT-4.1 as the judge. AlpacaEval2 and ArenaHardv2.0 report win rates, while WildBench and WritingBench report benchmark scores in their standard evaluation scales. All scores are reported such that higher is better. In each column, the best is in **bold**.

Supervision Method	Supervisor Size	Backbone	AlpacaEval2	Arena Hardv2.0	WildBench	Writing Bench
Baselines						
Native	-	Qwen3-8B	22.7	3.6	38.4	64.4
Native	-	Qwen3-32B	37.3	8.9	48.6	68.9
Scalar Reward Models						
Skywork-27B	27B	Qwen3-8B	40.2	6.8	45.2	67.6
Skywork-27B w/ responses	27B	Qwen3-8B	36.5	5.4	43.5	65.4
Nemotron-70B	70B	Qwen3-8B	23.0	3.2	38.1	62.3
Nemotron-70B w/ responses	70B	Qwen3-8B	24.6	3.8	39.6	65.0
Static Rubric						
RaR (Gunjal et al., 2025)	235B	Qwen3-8B	54.1	10.5	47.6	69.2
Dynamic Rubric						
DR-Generator-8B	8B	Qwen3-8B	**67.1**	**19.7**	**51.6**	**72.6**

F Robustness Experiments and Analysis

F.1 Robustness to Verifier Choice

DynamicRubric separates rubric generation from rubric verification: DR-Generator proposes weighted binary criteria, while DR-Verifier applies these criteria to candidate responses. The main experiments use Qwen3-30B-A3B as the verifier. To examine whether the gains depend on this specific verifier, we pair the same dynamic rubric generators with a range of frozen verifiers, including Llama-3.1-8B-Instruct and Qwen3 models of different scales. For each verifier, we compare Qwen3-8B zero-shot rubric generation, Qwen3-32B zero-shot rubric generation, and the trained DR-Generator-8B.

Table 6 shows that DR-Generator-8B remains the strongest rubric generator across all verifier choices. The absolute scores vary with verifier strength, as expected, but the relative pattern is stable: trained dynamic rubric generation consistently improves over both zero-shot generators under every verifier. This holds even when the verifier is a different model family, Llama-3.1-8B-Instruct, indicating that the learned rubric generator does not rely on a particular verifier or same-family verifier. These results support the modularity of DynamicRubric: training improves the quality of generated rubrics in a way that transfers across the frozen models used to verify them.

F.2 Robustness to LLM-as-Judge Choice

The main policy evaluation uses DeepSeek-V4-Flash as the automatic LLM-as-Judge. To test whether the observed policy gains depend on this judge, we re-evaluate the same policy checkpoints using GPT-4.1 while keeping the benchmark prompts, generated responses, and evaluation protocols unchanged. Table 7 shows that DynamicRubric remains the strongest supervision method under GPT-4.1 evaluation. It outperforms the native Qwen3-8B policy, scalar reward supervision, response-set-conditioned scalar variants, and static rubric supervision on all four benchmarks. Compared with the 235B static rubric supervisor, DynamicRubric improves AlpacaEval2, ArenaHardv2.0, WildBench, and WritingBench by 13.0, 9.2, 4.0, and 3.4 points, respectively. These results indicate that the downstream gains are robust to the choice of automatic judge.

Table 8: Evaluator performance of native and DynamicRubric-trained generators across Llama and Qwen backbones. Native denotes the corresponding backbone used as a zero-shot dynamic rubric generator. Acc. denotes strict top-1 accuracy, with tied top scores counted as incorrect. For each backbone and metric, the better result is in **bold**.

Benchmark	Metric	Llama-3.1-8B-Instruct		Qwen3-8B	
		Native	DR-Generator	Native	DR-Generator
JudgeBench	Acc.↑	34.8	**46.8**	36.4	**57.4**
Personalized-RB	Acc.↑	54.4	**71.1**	53.7	**73.3**
RM-Bench	Acc.↑	45.9	**50.2**	52.2	**69.8**
UltraFeedback	Acc.↑	56.2	**64.2**	54.7	**68.7**
AEOLLM	Acc.↑	19.2	**44.8**	17.8	**51.2**
LMSYS-Chat-1M	Acc.↑	13.3	**24.7**	12.0	**28.0**

Table 9: Policy performance of native and DynamicRubric-optimized policies across Llama and Qwen backbones. AlpacaEval2 and ArenaHardv2.0 report win rates, while WildBench and WritingBench report benchmark scores in their standard evaluation scales. All scores are reported such that higher is better. For each backbone and benchmark, the better result is in **bold**.

Benchmark	Llama-3.1-8B-Instruct		Qwen3-8B	
	Native	DR-Policy	Native	DR-Policy
AlpacaEval2	15.3	**28.7**	38.5	**67.1**
ArenaHardv2.0	0.4	**1.4**	9.0	**21.0**
WildBench	-17.1	**-7.7**	16.4	**30.7**
WritingBench	39.1	**47.5**	57.8	**64.3**

F.3 Cross-Backbone Robustness of DynamicRubric

To evaluate robustness to the choice of model backbone, we repeat the full DynamicRubric pipeline with Llama-3.1-8B-Instruct (Grattafiori et al., 2024). Both DR-Generator and DR-Policy are initialized from the native Llama-3.1-8B-Instruct checkpoint and trained with the same procedure used in the main experiments. We compare the resulting DynamicRubric-trained models against their native counterparts under the same evaluation protocols.

Table 8 shows that DynamicRubric training consistently improves the Llama-based evaluator. DR-Generator outperforms the native generator on every pairwise and listwise benchmark. This confirms that the evaluator-side benefit of response-set-conditioned rubric training is not specific to the Qwen backbone.

Table 9 shows that the Llama-based DR-Policy also improves over the native policy on all four open-domain generation benchmarks. The gains cover both win-rate benchmarks and benchmark-score evaluations, indicating that DynamicRubric supervision remains effective after changing the policy backbone. Together with the evaluator results, these experiments demonstrate cross-backbone robustness of the full bilevel DynamicRubric pipeline.

G Rubric Quality Meta-Evaluation

We further conduct a rubric-level meta-evaluation to examine whether DynamicRubric training improves the quality of generated rubric lists beyond their downstream ranking accuracy. We randomly sample 100 instances from Personalized-RewardBench and 100 instances from UltraFeedback. For each sampled instance, GPT-4.1 is given the user prompt, the response pool, and two anonymized weighted rubric lists generated before and after DynamicRubric training, respectively, by the native Qwen3-8B generator and the trained DR-Generator-8B. The display order of the two rubric lists is randomized for each instance. The judge compares the two rubric lists along task alignment, local discrimination, response awareness, specificity, weighting quality, and overall quality.

Table 10: Case-study evaluator scores and aggregate ranking metrics. Response A is the annotated preferred response. Seed-0 scores are raw weighted sums from one generated rubric set. Aggregate metrics are computed over five independently generated rubric sets with shuffled rubric orders. Tied top scores are counted as incorrect for Acc.

Generator	Response A	Response B	5-seed nDCG	5-seed Acc.
G_0	15	15	0.8155	0.0
G_2	53	24	1.0000	1.0

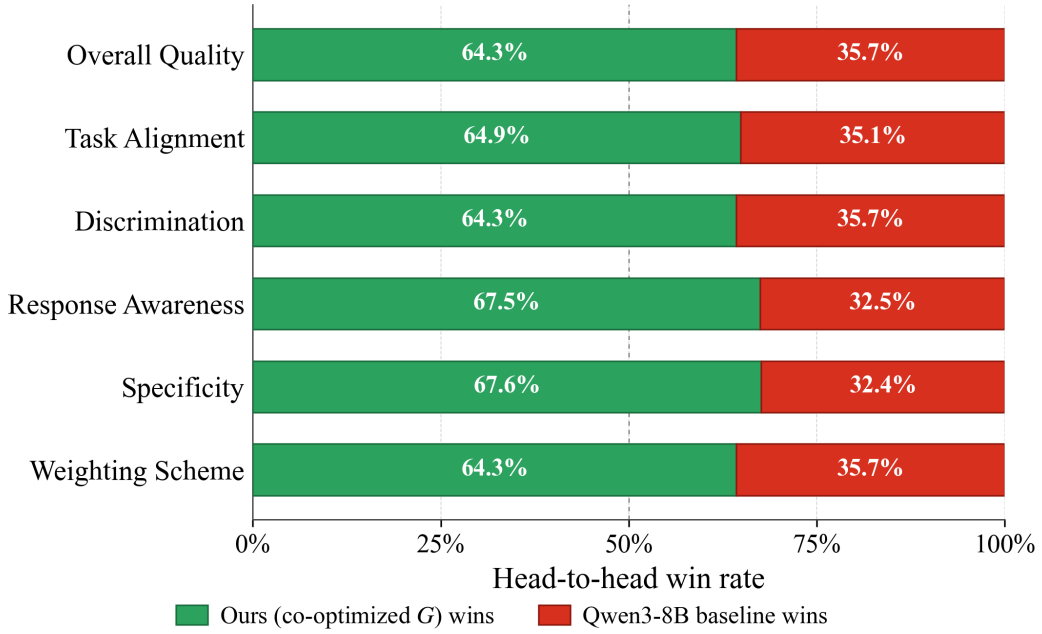


Figure 4: GPT-4.1 meta-evaluation of rubric quality. We report the pairwise win rate of DR-Generator-8B against the native Qwen3-8B generator on each rubric-quality dimension.

Figure 4 reports the fraction of pairwise comparisons in which GPT-4.1 prefers the DynamicRubric-trained generator. DR-Generator-8B is preferred over the native generator across all dimensions, with a 64.3% overall win rate. The strongest gains appear in response awareness (67.5%) and specificity (67.6%), indicating that DynamicRubric training makes rubric generation more sensitive to the decisive differences within the observed response set while maintaining concrete and assessable criteria. The judge also prefers DR-Generator-8B on task alignment (64.9%), local discrimination (64.3%), and weighting quality (64.3%). These results provide complementary qualitative evidence that DynamicRubric improves the rubric generator in the intended direction: generating criteria that are more discriminative, response-aware, and useful for evaluator-guided policy optimization.

H Case Study

We present a representative JudgeBench-GPT example to illustrate how DynamicRubric training changes response-set-conditioned rubric generation. Both generators receive only the user prompt and the unordered candidate response set. The annotated preference is withheld during rubric generation and used only for evaluation. The zero-shot Qwen3-8B generator G_0 and the DynamicRubric-trained generator G_2 are evaluated with the same frozen DR-Verifier. For each generated rubric set, DR-Verifier executes every rubric item as a binary yes/no check, and the weighted outcomes are aggregated into response-level evaluator scores. The score table reports one illustrative seed-0 run, while aggregate metrics are computed over five independently generated rubric sets with shuffled rubric orders.

Case-study input observed by both generators

```
Benchmark: JudgeBench-GPT

User prompt:
Estimate the angular momentum of Hurricane Florence at landfall, choose one option from A-J, and repeat the
      selected letter five times.

Candidate response set, unordered:

Response A:
Uses L = I omega with a disk-like moment of inertia.
Specifies numerical estimates for mass, radius, and angular velocity.
Obtains L about 3e19 kg m^2 / s and selects option B.

Response B:
Uses L = m r^2 omega.
Uses an angular velocity of about 1e-1 rad/s.
Obtains L about 1e23 kg m^2 / s and selects option A.
```

Rubrics generated by zero-shot Qwen3-8B generator G_0

```
1. The response provides the answer letter and repeats it five times.
2. The response includes step-by-step reasoning.
3. The response uses the correct angular-momentum formula.
4. The response stays focused on the question.
```

Representative rubrics generated by DynamicRubric-trained generator G_2

```
1. The response mentions the exact values used for mass, radius, or angular velocity.
2. The response compares the estimated angular momentum with the provided answer options.
3. The response mentions central pressure or wind speed as part of the estimation.
4. The response states the disk-like approximation for the hurricane.
5. The response discusses the proportionality constant in the moment of inertia.
6. The response computes angular velocity from tangential velocity and radius.
7. The response estimates mass and radius using typical hurricane-scale values.
8. The response computes both moment of inertia and angular momentum.
```

The zero-shot generator G_0 produces valid but weakly discriminative criteria. Its rubrics check answer formatting, step-by-step presentation, formula usage, and topical relevance. Both candidate responses satisfy these broad criteria: each gives an option in the required format, presents a calculation, invokes an angular-momentum formula, and stays on topic. The frozen DR-Verifier therefore assigns identical weighted scores to the two responses, which removes the unique top-ranked response and gives zero strict top-1 accuracy across all five seeds.

The DynamicRubric-trained generator G_2 produces criteria that better reflect the quality differences within the observed response set. Beyond checking formula presence, its rubrics examine whether the response specifies physically relevant quantities, estimates angular velocity from wind speed and radius, states an explicit moment-of-inertia model, and compares the resulting order of magnitude with the answer options. These criteria favor Response A, which carries out a coherent order-of-magnitude estimate, over Response B, whose final choice is driven by an implausibly large angular-velocity assumption. With the same frozen DR-Verifier, the generated rubric set induces a clear response-level score gap and ranks the annotated preferred response first in every seed.

This case illustrates the effect of learned dynamic rubrics: after DynamicRubric training, DR-Generator uses the observed alternatives to generate criteria that are both task-valid and locally discriminative for the current comparison.

I A Residual Analysis of Discriminability and Anchor Rewards

This appendix derives the expected relative score gap bound used in Section 3.4. The analysis separates the roles of the two Generator rewards. The discriminability reward encourages generated rubric items to separate responses in current-policy-induced response sets. The anchor reward calibrates the direction of these separations using ranked anchor data. The analysis does not require every generated rubric item to be aligned with the reference order or covered by the ordered-pair distribution; misalignment and insufficient coverage appear as explicit residual terms.

We first derive a residual lower bound that holds for arbitrary generated rubrics. Under an aggregate anchor-calibration condition, this residual bound yields constants $c_{\mathrm{disc}}, c_{\mathrm{anchor}} > 0$ and a round-dependent residual ϵ_t independent of ϕ such that

$$\Delta_{\mathrm{gap}}(E_\phi; \theta_{t-1}) \geq c_{\mathrm{disc}}\, \mathcal{J}^t_{\mathrm{disc}}(\phi; \theta_{t-1}) + c_{\mathrm{anchor}}\, \mathcal{J}^t_{\mathrm{anchor}}(\phi; \theta_{t-1}) + \epsilon_t. \tag{26}$$

Since ϵ_t is independent of ϕ within the evaluator-update candidate class, it does not affect the evaluator update.

Fix a prompt x and a conditioning response set

$$\mathcal{C} = \{y_1, \ldots, y_K\}. \tag{27}$$

Let

$$\mathcal{R}_\phi(x, \mathcal{C}) = \{(r_m, w_m)\}_{m=1}^M \tag{28}$$

be the rubrics generated from $\mathcal{C}$. For notational simplicity, we suppress the dependence of rubric items, weights, and verifier outcomes on $(x, \mathcal{C}, \phi)$ whenever x and $\mathcal{C}$ are fixed.

Let the normalized rubric weights be

$$a_m = \frac{w_m}{\sum_{\ell=1}^M w_\ell}, \qquad \sum_{m=1}^M a_m = 1. \tag{29}$$

For rubric item r_m, denote the DR-Verifier outcome on response y by

$$v_m(y) = V(x, y, r_m) \in \{0, 1\}. \tag{30}$$

The response-level evaluator induced by the rubric set is

$$E_\phi(x, y \mid \mathcal{C}) = \sum_{m=1}^M a_m v_m(y). \tag{31}$$

The discriminability reward can be written as

$$R_{\mathrm{disc}}(x, \mathcal{C}; \phi) = \sum_{m=1}^M a_m \bar{v}_m (1 - \bar{v}_m), \qquad \bar{v}_m = \frac{1}{K} \sum_{i=1}^K v_m(y_i). \tag{32}$$

Let $P(\cdot \mid x, \mathcal{C})$ be the ordered-pair distribution from Section 3.1, instantiated on the conditioning response set. For each rubric item m, define its signed gap contribution under P as

$$A_m = \mathbb{E}_{(y^+, y^-) \sim P(\cdot \mid x, \mathcal{C})} \left[v_m(y^+) - v_m(y^-) \right], \tag{33}$$

and define its split mass as

$$B_m = \mathbb{E}_{(y^+, y^-) \sim P(\cdot \mid x, \mathcal{C})} \left[|v_m(y^+) - v_m(y^-)| \right]. \tag{34}$$

Here, B_m measures how often rubric item m separates ordered pairs induced by $\mathcal{C}$, while A_m is the signed version that measures whether these separations follow the reference ordering.

Directional residual. For any $\gamma > 0$, define the directional residual of rubric item m as

$$\xi_m^\gamma = [\gamma B_m - A_m]_+, \tag{35}$$

where $[z]_+ = \max\{z, 0\}$, $z \in \mathbb{R}$. Then, by construction,

$$A_m \geq \gamma B_m - \xi_m^\gamma. \tag{36}$$

The residual ξ_m^γ measures the extent to which the split induced by rubric item m fails to follow the reference ordering on the evaluated ordered pairs.

Coverage residual. For any $\rho > 0$, define the aggregate coverage residual as

$$\zeta^{\rho}_{\mathrm{cov}}(x, \mathcal{C}; \phi) = \left[\rho R_{\mathrm{disc}}(x, \mathcal{C}; \phi) - \sum_{m=1}^{M} a_m B_m \right]_+ . \tag{37}$$

Then,

$$\sum_{m=1}^{M} a_m B_m \geq \rho R_{\mathrm{disc}}(x, \mathcal{C}; \phi) - \zeta^{\rho}_{\mathrm{cov}}(x, \mathcal{C}; \phi). \tag{38}$$

This residual is small when the ordered-pair distribution places nontrivial mass on response pairs separated by the generated rubric items.

Lemma 1: Fixed-input residual lower bound. For fixed x and $\mathcal{C}$ and any $\gamma, \rho > 0$,

$$\begin{aligned} &\mathbb{E}_{(y^+, y^-) \sim P(\cdot|x,\mathcal{C})} \left[E_\phi(x, y^+ \mid \mathcal{C}) - E_\phi(x, y^- \mid \mathcal{C}) \right] \\ &\geq \gamma \rho R_{\mathrm{disc}}(x, \mathcal{C}; \phi) - \sum_{m=1}^{M} a_m \xi^{\gamma}_m - \gamma \zeta^{\rho}_{\mathrm{cov}}(x, \mathcal{C}; \phi). \end{aligned} \tag{39}$$

Proof. Fix x, $\mathcal{C}$, and the generated rubric list. Define the expected evaluator gap over ordered pairs as

$$G_E = \mathbb{E}_{(y^+, y^-) \sim P(\cdot|x,\mathcal{C})} \left[E_\phi(x, y^+ \mid \mathcal{C}) - E_\phi(x, y^- \mid \mathcal{C}) \right]. \tag{40}$$

By the linearity of the evaluator in Equation 31,

$$\begin{aligned} G_E &= \sum_{m=1}^{M} a_m \mathbb{E}_{(y^+, y^-) \sim P(\cdot|x,\mathcal{C})} \left[v_m(y^+) - v_m(y^-) \right] \\ &= \sum_{m=1}^{M} a_m A_m. \end{aligned} \tag{41}$$

Using Equation 36 and $a_m \geq 0$,

$$G_E \geq \gamma \sum_{m=1}^{M} a_m B_m - \sum_{m=1}^{M} a_m \xi^{\gamma}_m. \tag{42}$$

Using Equation 38,

$$G_E \geq \gamma \rho R_{\mathrm{disc}}(x, \mathcal{C}; \phi) - \gamma \zeta^{\rho}_{\mathrm{cov}}(x, \mathcal{C}; \phi) - \sum_{m=1}^{M} a_m \xi^{\gamma}_m. \tag{43}$$

Substituting the definition of G_E in Equation 40 proves Lemma 1. □

Define the expected residual during the evaluator update at round t as

$$\eta^{t,\gamma,\rho}_{\mathrm{res}}(\phi) = \mathbb{E}_{x \sim \mathcal{D}} \mathbb{E}_{\mathcal{C} \sim \pi_{\theta_{t-1}}(\cdot|x)} \left[\sum_{m=1}^{M} a_m(x, \mathcal{C}) \xi^{\gamma}_m(x, \mathcal{C}) + \gamma \zeta^{\rho}_{\mathrm{cov}}(x, \mathcal{C}; \phi) \right]. \tag{44}$$

This quantity collects the residual direction error and the residual coverage error on the response-set distribution used for the evaluator update.

Aggregate anchor calibration. For the chosen $\gamma, \rho > 0$, suppose there exist a constant $\beta > 0$ and a round-dependent residual ϵ_t independent of ϕ within the evaluator-update candidate class at round t such that

$$-\eta^{t,\gamma,\rho}_{\mathrm{res}}(\phi) \geq \beta \, \mathcal{J}^{t}_{\mathrm{anchor}}(\phi; \theta_{t-1}) + \epsilon_t. \tag{45}$$

This condition states that the anchor ranking objective controls the aggregate residual left by discriminability-only training. For the Bradley–Terry-style anchor reward in Equation 15, $\mathcal{J}^{t}_{\mathrm{anchor}}$ is non-positive and increases toward zero as the induced score gaps better agree with the anchor ranking direction. Thus, increasing the anchor objective raises the lower bound by reducing the aggregate residual.

Theorem 1: Expected relative score gap guarantee. For any $\gamma, \rho > 0$, the expected relative score gap satisfies

$$\Delta_{\text{gap}}(E_\phi; \theta_{t-1}) \geq \gamma\rho \mathcal{J}^t_{\text{disc}}(\phi; \theta_{t-1}) - \eta^{t,\gamma,\rho}_{\text{res}}(\phi). \tag{46}$$

If the aggregate anchor calibration condition in Equation 45 holds, then there exist constants $c_{\text{disc}}, c_{\text{anchor}} > 0$ and a round-dependent residual ϵ_t independent of ϕ such that

$$\Delta_{\text{gap}}(E_\phi; \theta_{t-1}) \geq c_{\text{disc}}\, \mathcal{J}^t_{\text{disc}}(\phi; \theta_{t-1}) + c_{\text{anchor}}\, \mathcal{J}^t_{\text{anchor}}(\phi; \theta_{t-1}) + \epsilon_t. \tag{47}$$

In particular, one can take

$$c_{\text{disc}} = \gamma\rho, \qquad c_{\text{anchor}} = \beta. \tag{48}$$

Proof. Taking expectation in Lemma 1 over the evaluator-update sampling process, $x \sim \mathcal{D}$ and $\mathcal{C} \sim \pi_{\theta_{t-1}}(\cdot \mid x)$, the left-hand side becomes $\Delta_{\text{gap}}(E_\phi; \theta_{t-1})$. The first term on the right-hand side becomes

$$\gamma\rho \mathbb{E}_{x\sim\mathcal{D}} \mathbb{E}_{\mathcal{C}\sim\pi_{\theta_{t-1}}(\cdot|x)} \left[R_{\text{disc}}(x, \mathcal{C}; \phi)\right] = \gamma\rho \mathcal{J}^t_{\text{disc}}(\phi; \theta_{t-1}). \tag{49}$$

The remaining terms become $\eta^{t,\gamma,\rho}_{\text{res}}(\phi)$ by Equation 44. Therefore,

$$\Delta_{\text{gap}}(E_\phi; \theta_{t-1}) \geq \gamma\rho \mathcal{J}^t_{\text{disc}}(\phi; \theta_{t-1}) - \eta^{t,\gamma,\rho}_{\text{res}}(\phi), \tag{50}$$

which proves Equation 46. Applying the aggregate anchor calibration condition in Equation 45 gives

$$\Delta_{\text{gap}}(E_\phi; \theta_{t-1}) \geq \gamma\rho \mathcal{J}^t_{\text{disc}}(\phi; \theta_{t-1}) + \beta \mathcal{J}^t_{\text{anchor}}(\phi; \theta_{t-1}) + \epsilon_t. \tag{51}$$

Setting $c_{\text{disc}} = \gamma\rho$ and $c_{\text{anchor}} = \beta$ proves Equation 47. □

J Prompt Template

We provide our prompt templates here.

Prompt template for Policy.

```
Answer the user prompt.
Please reason step by step in <think>, write the complete user-facing answer in <answer>, and put the same
      final answer or conclusion from <answer> in minimal parseable form in <result>.

## User Prompt:
[user_prompt_replace]

## Output Format:
<think>
Your reasoning process.
</think>

<answer>
Your complete user-facing answer: include the final answer and the key supporting steps that justify it
</answer>

<result>
Your minimal final result or concise conclusion.
</result>
```

Prompt template for Generator.

```
You are an Evaluation Criteria Designer. Your task is to generate a list of binary ("yes"/"no") rubrics with
      weights to evaluate responses to a specific user prompt.

You will be provided with an unordered pool of actual model responses. Use this pool by comparing the
      responses to notice possible quality dimensions that matter for answering the user prompt. The response
       pool is only there to help you notice useful patterns and differences. If you notice a specific detail,
       example, entity, or mistake in the responses, first think about whether it reflects a broader quality
      dimension that is actually needed or meaningfully helpful for answering the user's request. If so,
      abstract it into a broader evaluative concept rather than copying it directly into a rubric. Do not
      treat the stronger responses in the pool as perfect; they may still contain errors or unnecessary
      details. Do not assume that every feature of stronger responses is equally necessary.
```

```
## Rubric Design Requirements:
1. Focus on the user's request:
   - Design rubrics to evaluate how well a response answers the user's actual request.
   - Use the response pool to notice which qualities help a response satisfy the user's request better.
   - Do not hardcode specific facts, numbers, names, entities, or other concrete details into your rubrics
     unless they are explicitly provided in the user prompt. Only use such details if you can abstract them
     into a broader evaluative concept that matters for answering the user's request.
   - Avoid rubrics that require external factual knowledge to verify.

2. Evaluate constraint adherence:
   - If the user prompt asks for specific formats, lengths, styles, or output structures, create rubrics to
     strictly check those constraints.

3. Atomicity and absolute binary (no "partially correct"):
   - Each rubric must evaluate exactly one distinct aspect.
   - Rubrics must be strictly binary (true or false).
   - If a response could reasonably satisfy only part of a rubric, that rubric is invalid and must be split
     or rewritten.
   - Bad example: "The response explains the cause and gives a solution."

4. Specificity and objectivity:
   - Prefer rubrics anchored to clear response features, explicit logical steps, or specific constraints
     requested by the user.
   - Make sure the boundary of the rubric is clear enough that different evaluators would usually agree on "
     yes" or "no".
   - Avoid vague descriptions.
   - Do not write rubrics around highly specific mistakes that appear in only one or two responses unless
     they reflect a broader quality dimension that would matter in other responses too.
   - Bad example: "The response is well-written and helpful."

5. Focus on substance:
   - Do not include rubrics that mainly distinguish responses by formatting or other presentation choices
     unless the user prompt explicitly requires them.
   - If the user did not ask for a specific format, a format-based rubric should usually be omitted.

6. Positive framing ("yes" = good):
   - Frame all rubrics so that a "yes" evaluation means the response is good, and "no" means it is bad.
   - Do not use double negatives.

7. No reference to specific responses:
   - Rubrics must not refer to any specific response.
   - Each rubric should be written as a general standard that can be applied to evaluate any response
     independently.

8. Mutually exclusive and self-contained:
   - Ensure each rubric evaluates a distinct aspect.
   - Each rubric should contain all the information needed to evaluate a response.
   - If one broad rubric and one narrower rubric are really checking the same thing, keep only one of them or
     rewrite them so they check clearly different things.
   - Do not treat two rubrics as overlapping just because they are related. Merge them only if a response
     would usually pass or fail them for the same reason.

9. Assigning weights (1-5 scale):
   - Assign an integer weight between 1 and 5 to each rubric based on how necessary that rubric is for a
     strong response to the user's request.
   - Essential (Weight 5): Core task requirements, critical user constraints, or safety-related requirements;
     if missing, the response is seriously flawed or invalid.
   - Important (Weight 3-4): Key reasoning, completeness, or clarity; these strongly affect response quality.
   - Optional (Weight 1-2): Nice-to-have improvements; these can distinguish an excellent response from a
     merely good one, but their absence should not make a response poor.
   - Do not assign a high weight merely because a rubric is common among stronger responses. A rubric can be
     useful for distinguishing stronger responses without being highly necessary.
   - Do not discard a rubric only because it is not essential. If it adds a different and useful quality
     dimension, keep it at a lower weight.
   - When the task supports it, include a mix of essential, important, and optional rubrics to help
     distinguish weaker, acceptable, and stronger responses.

10. The number of rubrics:
   - The appropriate number of rubrics depends on the task, but the rubric list should include enough
     independent rubrics to cover the meaningful quality dimensions needed for a discriminating evaluation.

## Rubric Design Workflow:
- First look at the user prompt on its own to identify the core task, any explicit constraints, and the
     minimum requirements for a response to count as a valid answer.
- Then consider the response pool together with the user prompt to identify the different quality dimensions
     that matter for this task.
- Use the response pool to notice useful quality differences, but do not let it override the user's actual
     request or constraints.
```

```
- When you notice a specific detail in the responses, abstract it into a broader quality dimension if that
      dimension is genuinely useful for evaluation.
- Decide for each quality dimension whether it is essential, important, or optional.
- After that, merge or remove rubrics only when they are really checking the same thing or when one adds
      almost no useful new information.
- Write each rubric as one complete declarative sentence.

## User Prompt:
[user_prompt_replace]

## Response List:
[response_list_replace]

## Output Format:
Please reason step by step, and then return only a valid JSON list inside <answer> and </answer> at the end
      of your output. The format of the <answer> block must be exactly:
<answer>
[
   {"rubric": "<rubric here>", "weight": <integer between 1 and 5>},
   {"rubric": "<rubric here>", "weight": <integer between 1 and 5>}
]
</answer>
```

Prompt template for Verifier.

```
You will be given:
1. A user prompt.
2. A list of rubrics, each with a unique ID.
3. A response to the user prompt.

Your task is to evaluate the response against each rubric independently and decide whether each rubric is
      satisfied.

## Scoring Rules:
- Answer "yes" if the response clearly meets the rubric.
- Answer "no" if the response does not meet the rubric. Partial matches should be scored as "no".
- If a rubric is poorly written and cannot be reasonably answered with "yes" or "no", it is a bad rubric -
      automatically judge it as "no".
- You must evaluate EVERY rubric provided in the list. Do not skip any.

## User Prompt:
[user_prompt_replace]

## Rubrics:
[rubrics_replace]

## Response:
[response_replace]

## Output Format:
Please return only a valid JSON list inside <answer> and </answer>. You MUST generate exactly one JSON object
       for EVERY rubric ID provided in the input. The number of items in your output list must exactly match
      the number of rubrics. Do not miss any ID, do not repeat any ID, and do not change the IDs. For each
      rubric, include all three fields: "id", "reason", and "judgement". Keep the same rubric ID order as the
       input list. Go through the IDs one by one. The "reason" should be brief and based only on what is
      clearly stated in the response. Do not guess missing facts, use outside knowledge, or revise your own
      answer inside the reason. If something is missing, simply say it is missing. Do not forget to write the
       "judgement" field for every item. The format must be exactly:
<answer>
[
   {"id": 1, "reason": "explain your judgement", "judgement": "yes"},
   {"id": 2, "reason": "explain your judgement", "judgement": "no"}
]
</answer>
Do not output anything else.
```