

# Reinforcement Learning with Robust Rubric Rewards

Ya-Qi Yu<sup>\*,†,✉</sup>, Hao Wang<sup>\*</sup>, Fangyu Hong<sup>\*</sup>, Xiangyang Qu<sup>\*</sup>,  
Gaojie Wu, Qiaoyu Luo, Nuo Xu, Huixin Wang, Wuheng Xu,  
Yongxin Liao, Zihao Chen, Haonan Li, Ziming Li, Dezhi Peng,  
Minghui Liao, Jihao Wu, Haoyu Ren, Dandan Tu

<sup>\*</sup> Core Contributors <sup>†</sup> Project Leader

Huawei Technologies Co., Ltd.

Project page: <https://github.com/yuyq96/rubrics>

## Abstract

While RLVR is effective for deterministically verifiable tasks, many vision-language tasks are only *partially verifiable*, demanding multi-criterion supervision. To bridge this gap, we propose *Reinforcement Learning with Robust Rubric Rewards* (RLR<sup>3</sup>), extending RL with criterion-level rewards. To ensure faithful scoring, RLR<sup>3</sup> routes each criterion to one of two execution paths: an LLM-as-an-extractor paired with a verifier, or an LLM-as-a-Judge for fuzzy criteria. We also introduce a minimal exposure strategy that masks verifier targets from extractors and source images from the GenRM. Furthermore, RLR<sup>3</sup> employs decoupled normalization to mitigate score saturation and hierarchical aggregation to prioritize essential criteria. Evaluated on Qwen3-VL-30B-A3B across 15 benchmarks, RLR<sup>3</sup> consistently outperforms RLVR, yielding a 4.7-point improvement over the base model and exceeding the official instruct-to-thinking model gap. Controlled audits show that deterministic verifier and minimal exposure strategy reduce exploitable false positives.

## 1 Introduction

Reinforcement Learning with Verifiable Rewards (RLVR) is a practical post-training approach that uses deterministic verifiers to provide unambiguous signals (Guo et al., 2025). It has been effective in math and code, where exact verifiers determine correctness (Shao et al., 2024; Team, 2025a; GLM, 2025), and has recently been extended to vision-language tasks such as OCR, counting, and grounding (Shen et al., 2025; Wang et al., 2025a; Team, 2026). However, this paradigm assumes the target is verifiable at the task level (Hong et al., 2025).

Many tasks are *partially verifiable*: a final answer may be checked exactly, whereas perceptual facts, reasoning steps, and instruction compliance

cannot always be verified deterministically. A single task-level signal collapses these distinctions into coarse supervision. This motivates instance-specific rubrics, which turn ambiguous response quality assessment into concrete and explicit evaluation criteria (Gunjal et al., 2025; Viswanathan et al., 2025; Huang et al., 2025; Yu et al., 2026; Pu et al., 2025; Xiong et al., 2025).

In online RL, rubric criteria must be scored accurately, as systematic execution errors create exploitable incentives for the policy. An intuitive optimization is to match each criterion with an appropriate execution path: verifiable criteria can be handled by prediction extraction followed by deterministic checking, while the others can be handled by semantic judgment.

We propose *Reinforcement Learning with Robust Rubric Rewards* (RLR<sup>3</sup>), extending RL training with robust criterion-level rewards. As shown in Figure 1, RLR<sup>3</sup> routes each rubric criterion to a text-only LLM extractor followed by a deterministic verifier, falling back to an LLM-as-a-Judge only when deterministic checking is unavailable. To prevent potential shortcutting, verifier targets are hidden from the extractor, and source images are masked from both paths. Furthermore, RLR<sup>3</sup> remaps saturated scores within rollout groups, prioritizes critical over supplementary criteria via hierarchical aggregation, and trains the Generative Reward Model (GenRM) using RLVR.

By making verifiability a criterion-level property, RLR<sup>3</sup> supports both fully and partially verifiable tasks in a single GRPO loop. Evaluated on Qwen3-VL-30B-A3B, RLR<sup>3</sup> improves the macro average over RLVR across 3 open-source training mixtures: 76.4 to 77.7 on ViRL, 76.4 to 78.1 on OpenMMR, and 77.4 to 78.2 on DeepVision. Reward model audits further show that deterministic verification and minimal exposure reduce false positives on failure responses without harming scoring accuracy, while the RLVR-trained GenRM reaches 95.0%

<sup>✉</sup>E-mail: yuyaqi5@huawei.com

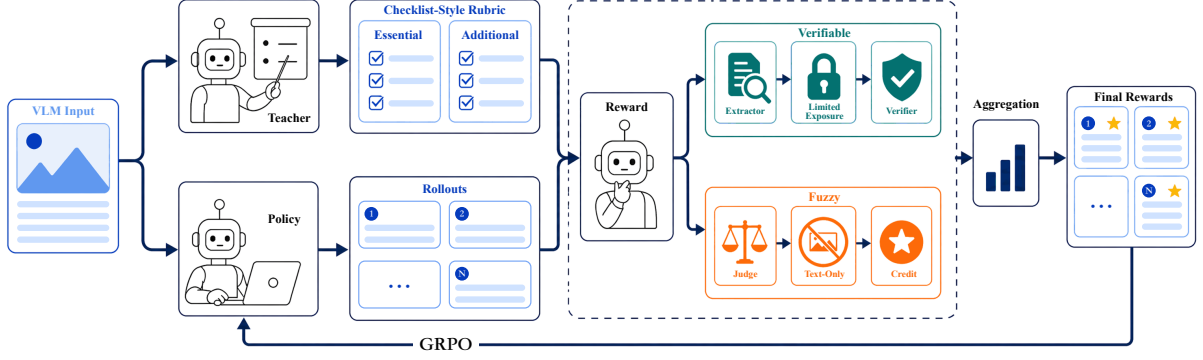


Figure 1: **Overview of RLR<sup>3</sup>**. Instance-specific rubrics turn response quality assessment into concrete criteria and criterion-level rewards. Verifiable criteria are routed to a text-only LLM-as-an-extractor followed by deterministic verification, while fuzzy criteria are routed to a text-only LLM-as-a-Judge.

criterion-level accuracy on the held-out test set.

Our contributions are summarized as follows:

- We identify multi-criteria rubric-based RL as a natural fit for partially verifiable tasks.
- We propose RLR<sup>3</sup>, a robust criterion-level reward framework that routes verifiable criteria to LLM extraction plus deterministic verifier and fuzzy criteria to LLM judgment, both under a minimal exposure strategy.
- We improve signal resolution and reliability through decoupled normalization and hierarchical aggregation for multi-criteria.
- We validate RLR<sup>3</sup> through policy optimization comparisons with RLVR, GenRM reliability evaluation, and failure-mode audits.
- An experiment with a shared initialization for the policy, rubric generator, and GenRM suggests a unified, co-evolving framework.

## 2 Related Works

**RLVR.** RLVR uses deterministic verifiers to provide precise rewards (Guo et al., 2025). While initially applied to math and code via exact matches or unit tests (Shao et al., 2024; Team, 2025a; GLM, 2025), this paradigm has expanded to multimodal tasks such as OCR, counting, and grounding (Shen et al., 2025; Wang et al., 2025a; Team, 2026). Despite its transparency, the applicability of RLVR fundamentally depends on the availability of verifiable ground truth (Hong et al., 2025).

**Rubrics.** Rubric-based evaluation improves interpretability and granularity by decomposing response quality into explicit criteria. In NLP, expert rubrics are used to evaluate complex capabilities such as open-ended generation and professional reasoning (Hashemi et al., 2024; Starace et al., 2025; Arora et al., 2025; Wang et al., 2025b;

Akyürek et al., 2025), and have been extended to post-training (Gunjal et al., 2025; Viswanathan et al., 2025; Huang et al., 2025; Zhou et al., 2025; Xu et al., 2026). Recent work has explored rubric generation through direct synthesis or pairwise elicitation (Liu et al., 2025; Rezaei et al., 2025; Xie et al., 2025; Li et al., 2026). In multimodal settings, rubrics support both evaluation (Pu et al., 2025; Xiong et al., 2025) and alignment (Kong et al., 2026; Yu et al., 2026). RLR<sup>3</sup> complements these directions by focusing on criterion-level reward execution during online RL, with distinct execution paths for verifiable and fuzzy criteria.

## 3 Preliminaries

We optimize the policy with Group Relative Policy Optimization (GRPO) (Shao et al., 2024; Yu et al., 2025) under a strict on-policy setting without a KL penalty. For each input  $x$ , the current policy  $\pi_\theta$  samples a group of  $G$  responses  $\{y_1, \dots, y_G\}$ . Each sampled response  $y_i$  is assigned a scalar reward  $\tilde{r}_i$ . If a response exceeds the maximum length, its reward is set to 0. We maximize the following objective:

$$\mathcal{J}(\theta) = \mathbb{E} \left[ \frac{\sum_{i=1}^G \sum_{t=1}^{|y_i|} \frac{\pi_\theta(y_{i,t}|x, y_{i,<t})}{\text{sg}(\pi_\theta(y_{i,t}|x, y_{i,<t}))} \hat{A}_i}{\sum_{i=1}^G |y_i|} \right], \quad (1)$$

where  $\text{sg}(\cdot)$  denotes the stop-gradient operator and  $\hat{A}_i = (\tilde{r}_i - \text{mean}(\{\tilde{r}_j\}_{j=1}^G)) / \text{std}(\{\tilde{r}_j\}_{j=1}^G)$  is the group-relative advantage. Section 4 defines how the scalar reward  $\tilde{r}_i$  is constructed, including the rubric, its criterion execution paths, and the aggregation procedure.

## 4 Methodology

To ensure accurate and robust rewards, RLR<sup>3</sup> follows three principles. First, it prioritizes verifiability to limit opportunities for exploitation (Section 4.1). Second, it restricts evidence exposure to prevent unintended shortcuts (Section 4.2). Third, it preserves criterion-level distinctions during aggregation (Section 4.3).

### 4.1 Rubric Generation

For an input  $x$ , we define the instance-specific rubric as  $C^x = \{c_1^x, \dots, c_K^x\}$ , where each criterion is represented as  $c_k^x = \langle d_k^x, t_k^x, w_k^x, V_k, z_k^x \rangle$ . In this quintuplet,  $d_k^x$  is the criterion description,  $t_k^x \in \{\text{ESSENTIAL}, \text{ADDITIONAL}\}$  is the criterion type,  $w_k^x$  is a non-negative weight,  $V_k$  is the verifier tag, and  $z_k^x$  is the associated reference object. Specifically, the verifier tag determines the execution path of each criterion:

- **Verifiable criteria** ( $V_k \neq \emptyset$ ) are used when the relevant content can be extracted from the response and checked deterministically against ground-truth targets. In this case,  $z_k^x$  stores the target arguments passed to the corresponding verifier function, for example when the response must provide an option letter or a specific time string.
- **Fuzzy criteria** ( $V_k = \emptyset$ ) are used when judgment needs semantic understanding and cannot be reduced to deterministic matching. In this case,  $z_k^x$  stores the textual reference used by the LLM-as-a-Judge, for example whether the response conveys the same meaning as the reference using different wording or follows a specific instruction.

**Verifier library.** We use a compact verifier library that covers common value types, including plain text, expressions, times, lists, bounding boxes, and points. During rubric generation, the rubric generator is given only the verifier tags allowed for the current task, together with the intended use, input schema, and examples for each verifier. For example, OCR tasks expose the text verifier, while grounding tasks expose the bounding-box or point verifier. This restriction prevents verifier misuse. It also rules out cases in which a complex semantic criterion is recast as a Boolean judgment and checked with the expression verifier. For example, “the response correctly states that the man is wearing a red hat and standing to the left of the bicycle.” This design ensures that verifiers check only values

extracted from a response, rather than judgments produced by the extractor.

**Proprietary rubric generators.** First, we use a two-stage multi-teacher aggregation pipeline to improve zero-shot rubric generation: Gemini-3.0-Pro and Gemini-3.0-Flash each draft candidate criteria independently. Gemini-3.0-Pro then aggregates the drafts into the final rubrics. Appendix A.1 describes the rubric schema. Appendices B.1 and B.2 detail the generation pipeline.

**Open-source rubric generators.** We also study an open-source approach in which a fine-tuned model produces finalized rubrics in one pass, replacing both drafting and aggregation. In our main open-source setting, the rubric generator, policy, and reward model are all initialized from the same open-source checkpoint, i.e., Qwen3-VL-30B-A3B-Instruct.

### 4.2 Criterion Execution

We introduce distinct execution paths for verifiable and fuzzy criteria. Both paths follow a minimal-exposure principle. For verifiable criteria, if the target arguments  $z_k^x$  were visible, the extractor might copy the required values from them instead of extracting them from the response. We therefore withhold  $z_k^x$  from the extractor on the verifiable path. Another failure mode could arise if the source image were visible. The judge might infer the task even when the response does not state it. To avoid this shortcut, both execution paths adopt an LLM sharing the same parameters  $\phi$ .

#### 4.2.1 Verifiable: LLM-as-an-Extractor

Let  $x = (x^t, x^i)$ , where the input is decomposed into a text prompt  $x^t$  and a source image  $x^i$ . For verifiable criteria, the extractor generates criterion-level reasoning and an extracted value  $(\eta_k, \hat{a}_k) = E(x^t, y, d_k^x, V_k; \phi)$ , where  $E(\cdot; \phi)$  uses the shared LLM parameters  $\phi$ . The deterministic verifier then computes  $s_k = V_k(\hat{a}_k, z_k^x)$ . For a time verifier,  $z_k^x$  includes the time string and string format. Appendix C summarizes the verifier specifications used in our implementation.

Under minimal exposure, the extractor only sees the text prompt  $x^t$ , the response  $y$ , the criterion description  $d_k^x$ , and the verifier tag  $V_k$ . Hence, it must identify the value from the response  $y$  rather than copying it from  $z_k^x$ .

### 4.2.2 Fuzzy: LLM-as-a-Judge

For fuzzy criteria in the reference-grounded setting, the judge excludes the source image and predicts criterion-level reasoning alongside a discrete score, formulated as  $(\eta_k, s_k) = J(x^t, y, d_k^x, z_k^x; \phi)$ . Following the scoring scheme in rDPO (Yu et al., 2026),  $s_k \in \{0, 0.5, 1\}$  maps to no, partial, and full credit, respectively.

### 4.2.3 Structured Outputs

These paths together produce a single JSON object containing a global reasoning field and a list of criterion records. Appendix A.2 lists the full schema, which makes reward execution easy to review and simplifies the interface between rubric generation, execution, and RL training.

## 4.3 Reward Aggregation

### 4.3.1 Decoupled Normalization

For an input  $x$  and a response group  $\{y_i\}_{i=1}^G$ , let  $s_{k,i}$  denote the raw score assigned to response  $y_i$  under criterion  $c_k^x$ . Verifier scores may span different, often narrow ranges, causing direct aggregation to lose fine-grained information.

We therefore apply criterion-level score normalization before aggregation. Naive min-max normalization can, however, inflate a uniformly poor group by mapping its best response to full credit. We use a threshold  $\tau$  to constrain the normalization, preserving absolute failure information while improving relative resolution. Given a threshold  $\tau$ , let  $s_{k,\min} = \min_i s_{k,i}$  and  $s_{k,\max} = \max_i s_{k,i}$ . We define the group-wise lower bound  $\ell_k$  and upper bound  $u_k$  as:

$$\ell_k = \begin{cases} 0, & s_{k,\min} < \tau, \\ 0.5, & s_{k,\min} \geq \tau, \end{cases} \quad u_k = \begin{cases} 1, & s_{k,\max} > \tau, \\ 0.5, & s_{k,\max} \leq \tau. \end{cases} \quad (2)$$

We then define

$$\tilde{s}_{k,i} = \begin{cases} u_k, & s_{k,\min} = s_{k,\max} > \tau, \\ \ell_k, & s_{k,\min} = s_{k,\max} \leq \tau, \\ \frac{(s_{k,i} - s_{k,\min})(u_k - \ell_k)}{s_{k,\max} - s_{k,\min}} + \ell_k, & \text{otherwise.} \end{cases} \quad (3)$$

### 4.3.2 Hierarchical Aggregation

The base content reward is computed as  $r = \sum_{k=1}^K w_k^x \tilde{s}_k$ . This weighted sum captures fine-grained differences across responses, but by itself it treats all gains as mutually compensatory. As a result, strong performance on additional criteria could offset failures on essential ones, even though the rubric is meant to prioritize the latter. We therefore gate the base reward by criterion type, so that

additional criteria refine the score only after the essential criteria are satisfied. We adopt a consistent scoring convention, treating scores below 0.5 as failures and scores in  $[0.5, 1)$  as partial satisfaction. The corresponding content mask is defined by:

$$m_c = \begin{cases} 0, & |\{k : t_k^x = \text{Ess.}, \tilde{s}_k < 0.5\}| \geq 1, \\ 0, & |\{k : t_k^x = \text{Ess.}, 0.5 \leq \tilde{s}_k < 1\}| \geq 2, \\ 1, & \text{otherwise.} \end{cases} \quad (4)$$

Some response-level violations, such as repetition loops or language mixing, are also handled as hard format constraints. We define a binary format mask  $m_f \in \{0, 1\}$ , which is set to 0 when any such violation is triggered and to 1 otherwise. The final reward is  $\tilde{r} = m_c \cdot m_f \cdot r$ .

## 4.4 GenRM RL

Reward robustness depends not only on rubric design, but also on reliable rubric execution. Since moderately-sized LLMs often struggle with complex instructions and multi-field structured outputs, we train the GenRM with RLVR before using it in policy optimization.

We prepare the GenRM data by sampling candidate responses and scoring them using multiple frontier models that independently execute the target rubric. For each criterion, we retain the median score across teachers. For verifiable criteria, we additionally extract the output value from a teacher whose score matches this median. This multi-teacher ensemble effectively mitigates single-model execution noise.

Training adopts two verifiable rewards:

- **Format reward**, which is 1 if the output is a valid JSON object and all required fields have the expected types, otherwise 0.
- **Content reward**, which is the average accuracy over criteria. For verifiable criteria, we match the extracted values. For fuzzy criteria, we match the predicted credit.

## 4.5 Policy RL

The preceding components together define a fine-grained reward used to train the policy via GRPO. For each input  $x$ , we sample a group of responses  $\{y_i\}_{i=1}^G$ , score them using the rubrics, and normalize the final rewards  $\tilde{r}_i$  within the group. Consequently, both fully and partially verifiable tasks are optimized within a unified GRPO loop, eliminating the need to split RLR<sup>3</sup> into separate RLVR and general RL stages.

Table 1: **Statistics of the training corpora after preprocessing.** “Raw” denotes the original training split, “De-dup.” denotes the de-duplicated split, and “Filtered” denotes the subsets retained by the “Any” and “Essential” filtering rules. The “Essential” filtering rule is adopted for policy training. “MCQ Ratio” reports the fraction of multiple-choice questions. “Initial Reward” is the average rubric score of base-model rollouts on all of the converted instances and on the filtered subset.

| Dataset    | # Raw   | # De-dup. | # Filtered |           | MCQ Ratio |           |          | Initial Reward |          |
|------------|---------|-----------|------------|-----------|-----------|-----------|----------|----------------|----------|
|            |         |           | Any        | Essential | De-dup.   | Converted | Filtered | Converted      | Filtered |
| ViRL       | 38,870  | 38,870    | 16,444     | 9,551     | 31.8%     | 8.8%      | 13.7%    | 0.8156         | 0.4249   |
| OpenMMR    | 74,971  | 74,145    | 36,627     | 26,070    | 22.1%     | 6.5%      | 8.4%     | 0.7397         | 0.3556   |
| DeepVision | 103,484 | 92,491    | 63,549     | 55,550    | 48.4%     | 13.2%     | 15.2%    | 0.4713         | 0.3147   |

**RLVR as a special case.** When every criterion is verifiable, the reward is obtained solely by aggregating verifier scores. In the single-criterion case, we have  $\tilde{r} = V(E(y), z^x)$ , where  $E$  denotes a rule-based extractor and  $V$  verifies the extracted value against  $z^x$ . In this case, RLR<sup>3</sup> reduces to RLVR.

## 5 Experiments

### 5.1 Experimental Setup

**Data.** We use the training splits of three open-source corpora as shared training sources for both methods: ViRL (Wang et al., 2025a), OpenMMR (Zhang et al., 2025), and DeepVision (Sun et al., 2026). We further convert a subset of multiple-choice questions into open-ended questions when a VLM judges that the question remains well-posed without the answer options and still admits a unique correct answer. We remove duplicates at the image-question level and pair each remaining instance with an instance-specific rubric containing both verifiable and fuzzy criteria, following Section 4.1. We then perform filtering by sampling 8 rollouts from the base model and retaining only examples on which at least one rollout receives no credit on any criterion or on an essential criterion, depending on the target subset. Table 1 summarizes the resulting training data.

**Training.** For policy initialization, we adopt Qwen3-VL-30B-A3B-Instruct (Team, 2025b) as the base model. Models are trained using Adam with a constant learning rate of  $1 \times 10^{-6}$ , a weight decay of 0.01,  $\beta_1 = 0.9$ , and  $\beta_2 = 0.999$ . The maximum prompt and response lengths are 2K and 6K tokens, respectively. For DeepVision, we increase them to 4K and 12K. Training uses a global batch size of 128 and 8 rollouts for 1,000 steps. All reported results use the checkpoint at 1,000 training steps. Appendix F summarizes the computational cost of rubric execution during policy training.

**Benchmarks.** We evaluate models on official test or testmini splits of 15 public benchmarks across math, general VQA, counting, and document VQA: We-Math (Qiao et al., 2025), DynaMath (Zou et al., 2025), MathVision (Wang et al., 2024a), MathVerse (Zhang et al., 2024), MathVista (Lu et al., 2023), MMMU-Pro (Yue et al., 2025), RealWorldQA (xAI, 2024), MMStar (Chen et al., 2024), SimpleVQA (Cheng et al., 2025), CountBenchQA (Beyer et al., 2024), InfoVQA (Mathew et al., 2022), DocVQA (Mathew et al., 2020), ChartQA (Masry et al., 2022), and CharXiv (DQ/RQ) (Wang et al., 2024b). We prompt the model to think step-by-step and enclose the final answer in a boxed span. Evaluation employs rule-based matching for structured outputs (e.g., letters, numbers, and expressions) and GPT-4o-mini as a judge for open-ended outputs.

### 5.2 Main Results

We compare RLVR and RLR<sup>3</sup> across the three training mixes, as reported in Table 2.

**RLR<sup>3</sup> improves average performance over RLVR.** Specifically, on the macro average, RLR<sup>3</sup> improves over RLVR from 76.4 to 77.7 on ViRL, from 76.4 to 78.1 on OpenMMR, and from 77.4 to 78.2 on DeepVision. At the benchmark level, RLR<sup>3</sup> attains higher scores than RLVR on most benchmarks across the three training mixes. Representative gains appear on We-Math, MathVision, MathVerse, MMMU-Pro, and CountBenchQA, with especially large margins on OpenMMR We-Math (67.3  $\rightarrow$  74.3), DeepVision MathVerse (73.9  $\rightarrow$  79.6), and ViRL CountBenchQA (87.3  $\rightarrow$  93.2). On the benchmarks where RLR<sup>3</sup> does not improve over RLVR, the gaps are usually small.

**RLR<sup>3</sup> achieves larger RL gains than the official Thinking variant.** Our evaluated base model reaches a macro average of 73.5, compared with

Table 2: **Performance comparison on visual benchmarks.** In the RLR<sup>3</sup> columns, green highlights improvement over RLVR and red highlights degradation. “Official” and “GPT-5 mini” denotes results from the Qwen3-VL technical report. “Base” denotes our own evaluation of the instruct checkpoint.

| Benchmark        | Base     | ViRL        |                  | OpenMMR     |                  | DeepVision  |                  | Official    |          | GPT-5 mini  |         |
|------------------|----------|-------------|------------------|-------------|------------------|-------------|------------------|-------------|----------|-------------|---------|
|                  | instruct | RLVR        | RLR <sup>3</sup> | RLVR        | RLR <sup>3</sup> | RLVR        | RLR <sup>3</sup> | thinking    | instruct | high        | minimal |
| We-Math          | 56.8     | 68.3        | 70.4             | 67.3        | <b>74.3</b>      | 72.9        | <u>73.6</u>      | 70.0        | 56.9     | 70.2        | 51.4    |
| DynaMath         | 74.8     | 74.1        | 78.4             | 78.7        | 78.0             | 80.0        | 76.4             | <u>80.1</u> | 73.4     | <b>81.4</b> | 71.3    |
| MathVision       | 63.1     | 61.4        | 63.4             | 63.1        | 66.1             | 65.5        | <u>68.8</u>      | 65.7        | 60.2     | <b>71.9</b> | 46.6    |
| MathVerse (mini) | 71.7     | 72.1        | 76.6             | 75.5        | 76.9             | 73.9        | <b>79.6</b>      | <b>79.6</b> | 70.2     | 78.8        | 36.5    |
| MathVista (mini) | 79.8     | 82.3        | <u>83.7</u>      | 83.5        | <b>83.9</b>      | 83.0        | 82.8             | 81.9        | 80.1     | 79.1        | 59.6    |
| MMMU-Pro         | 64.2     | 64.5        | 66.8             | 64.1        | 66.6             | 66.8        | <b>67.4</b>      | 63.0        | 60.4     | <u>67.3</u> | 53.7    |
| RealWorldQA      | 74.4     | <u>77.9</u> | 76.7             | 76.7        | 77.6             | 76.6        | 75.6             | 77.4        | 73.7     | <b>79.0</b> | 73.3    |
| MMStar           | 74.1     | 75.7        | 76.9             | <b>77.8</b> | 76.9             | <u>77.1</u> | 76.6             | 75.5        | 72.1     | 74.1        | 61.3    |
| SimpleVQA        | 49.9     | 54.0        | 53.4             | 52.9        | <u>54.6</u>      | 52.1        | 52.4             | 54.3        | 52.7     | <b>56.8</b> | 50.3    |
| CountBenchQA     | 88.7     | 87.3        | <b>93.2</b>      | 89.7        | 90.3             | 88.3        | <u>92.6</u>      | 90.0        | 89.8     | 91.0        | 84.1    |
| InfoVQA          | 86.7     | 89.1        | 89.3             | 87.3        | <b>90.2</b>      | <u>90.2</u> | 89.5             | 85.6        | 81.8     | 77.6        | 72.8    |
| DocVQA           | 94.5     | 95.1        | 95.1             | 94.3        | <u>95.2</u>      | 94.9        | 95.0             | <b>95.5</b> | 95.0     | 90.5        | 90.6    |
| ChartQA          | 88.6     | 90.6        | <b>91.1</b>      | 90.6        | 90.6             | <u>91.0</u> | 90.3             | 89.4        | 86.8     | 57.5        | 57.8    |
| CharXiv (DQ)     | 81.7     | <u>90.5</u> | 89.8             | 85.3        | 87.5             | 88.2        | <b>91.3</b>      | 86.9        | 85.5     | 89.4        | 78.6    |
| CharXiv (RQ)     | 53.4     | 62.4        | 61.4             | 58.4        | <u>62.7</u>      | 61.2        | 61.8             | 56.6        | 48.9     | <b>68.6</b> | 48.9    |
| Macro Average    | 73.5     | 76.4        | 77.7             | 76.4        | <u>78.1</u>      | 77.4        | <b>78.2</b>      | 76.8        | 72.5     | 75.5        | 62.5    |

72.5 for the officially reported instruct results, a small gap of 1.0 point. Our best RLR<sup>3</sup> model improves from 73.5 to 78.2, a gain of 4.7 points, which is larger than the officially reported 4.3-point gain from the instruct model to the thinking model.

The OpenMMR results highlight a limitation: while RLR<sup>3</sup> still outperforms RLVR on the macro average, its margin is smaller than on ViRL and DeepVision. This is likely because OpenMMR features complex visual puzzles, where an automatic rubric generator relying solely on outcome-level ground truth offers limited supervision. Consequently, a promising future direction is the synthetic construction of visual puzzles with both outcome labels and fine-grained perceptual annotations to enable more informative rubric execution.

### 5.3 Open-Source Rubric Generator

We evaluate whether an open-source model can replace the costly proprietary pipeline at rubric-generation time. We report results on the ViRL training mix, with the generator preparation details provided in Appendix E.

**Open-source rubric generator can approach proprietary-pipeline performance.** In Table 3, the Instruct generator achieves a macro average

Table 3: **Comparison of open-source rubric generator with ViRL training mix.** We report macro-average performance across 15 benchmarks.

| Method           | Rubric Generator          | Avg. |
|------------------|---------------------------|------|
| RLVR             | –                         | 76.4 |
| RLR <sup>3</sup> | Gemini-3.0-Pro/Flash      | 77.7 |
| RLR <sup>3</sup> | Qwen3-VL-30B-A3B-Instruct | 77.2 |
| RLR <sup>3</sup> | Qwen3-VL-30B-A3B-Thinking | 75.8 |

of 77.2, outperforming RLVR by 0.8 points while remaining only 0.5 points below the proprietary pipeline. This gain over RLVR is particularly notable because the rubric generator and policy originate from the same model initialization: *effective rubric supervision does not necessarily require a generator built on a stronger base model than the policy*. The proprietary pipeline, however, operates with a substantially larger inference-time compute budget, including the reasoning tokens consumed across multiple independent drafting and aggregation calls. This motivates the Thinking variant as a way to introduce greater deliberation within the open-source generator. Nevertheless, the Thinking variant reaches only 75.8, falling behind both RLVR and the Instruct variant. We hypothesize

Table 4: **Policy-level ablations on ViRL39K.** We report macro-average performance across 15 benchmarks after 1,000 policy-training steps.  $\Delta$  is measured relative to the full RLR<sup>3</sup> configuration.

| Variant                     | Avg. | $\Delta$ |
|-----------------------------|------|----------|
| Full RLR <sup>3</sup>       | 77.7 | —        |
| w/o Hierarchical Gating     | 76.6 | −1.1     |
| w/o Criterion Weight        | 77.3 | −0.4     |
| w/o Decoupled Normalization | 76.7 | −1.0     |

that supervised imitation of substantially rewritten reasoning trajectories introduces a mismatch with the model’s native reasoning behavior. Training the Thinking generator with RL, using finalized rubric quality as direct feedback, offers a more promising route to surpassing the Instruct generator and approaching the proprietary pipeline.

**Shared initialization points toward a unified RLR<sup>3</sup> implementation.** These results also highlight a broader structural opportunity. In this setting, the rubric generator, policy, and GenRM are all initialized from the same Qwen3-VL-30B-A3B-Instruct checkpoint, demonstrating that the three roles can be instantiated from a common initialization without giving up most of the gains provided by the proprietary rubric pipeline. This common foundation offers an early glimpse of a fully unified RLR<sup>3</sup> implementation, in which the three components could remain parameter-tied throughout training and specialize through role-specific prompts and optimization objectives. Joint optimization could further enable the three roles to co-evolve: better rubrics would provide sharper supervision, a stronger GenRM would translate that supervision into more reliable rewards, and an improved policy would produce harder and more informative responses for refining both components.

#### 5.4 Policy-Level Ablations

**Hierarchical aggregation preserves rubric priorities.** Table 4 shows that removing any policy-level component reduces the macro average. Removing hierarchical gating produces the largest drop, demonstrating the importance of preventing gains on additional criteria from compensating for failures on essential requirements. Criterion weights complement this hard constraint by encoding finer-grained differences in criterion importance. Using uniform weights reduces the average by 0.4 points.

**Decoupled normalization improves reward**

**resolution.** Removing it decreases the average by 1.0 point, showing that policy training benefits from preserving distinctions among saturated criterion scores. This effect may become more pronounced on perception-dominated tasks, where non-binary criteria constitute a larger share of the reward and exhibit more heterogeneous score ranges.

#### 5.5 Long-Term Training Stability

To study the long-term stability and scaling behavior of both methods, we plot their performance curves over training in Figure 2.

**RLR<sup>3</sup> remains strong over a broader range of checkpoints than RLVR.** Across many benchmarks, RLVR improves in the early stage of training and then plateaus or degrades, whereas RLR<sup>3</sup> usually maintains or further improves its performance over a longer portion of the trajectory. The best performance of RLR<sup>3</sup> is also higher than that of RLVR on most benchmarks. This suggests that finer-grained and more robust reward modeling can stabilize RL training and make it easier to scale.

This trajectory-level behavior also suggests a promising future direction. These gains are not yet fully consolidated into a single final model. Future work could explore On-Policy Distillation (OPD) to better transfer the strengths of multiple experts into one stronger policy.

#### 5.6 GenRM Reliability

We construct the GenRM training and validation dataset by sampling policy inference results, scoring responses with multiple frontier models, and using majority-voted criterion-level labels. The dataset is balanced between perfect and imperfect responses and carefully split to ensure no input overlap between splits. Table 5 summarizes the resulting dataset. For evaluation, we verify GenRM’s JSON outputs against the dataset labels, focusing on schema validity, criterion-slot matching, and function-calling integrity. Argument accuracy is assessed using verifiers rather than exact matching.

We compare the base model and two fine-tuned variants. Table 6 shows that the base model follows the coarse schema reasonably well, but remains unreliable once the output must be executable and correctly scored. Its schema and criterion-slot accuracies are 98.6% and 98.7%, while execution accuracy drops to 82.0% and credit accuracy to 77.0%. SFT solves most instruction-following errors, reaching 100.0% accuracy on all three format metrics. Compared with this baseline, the

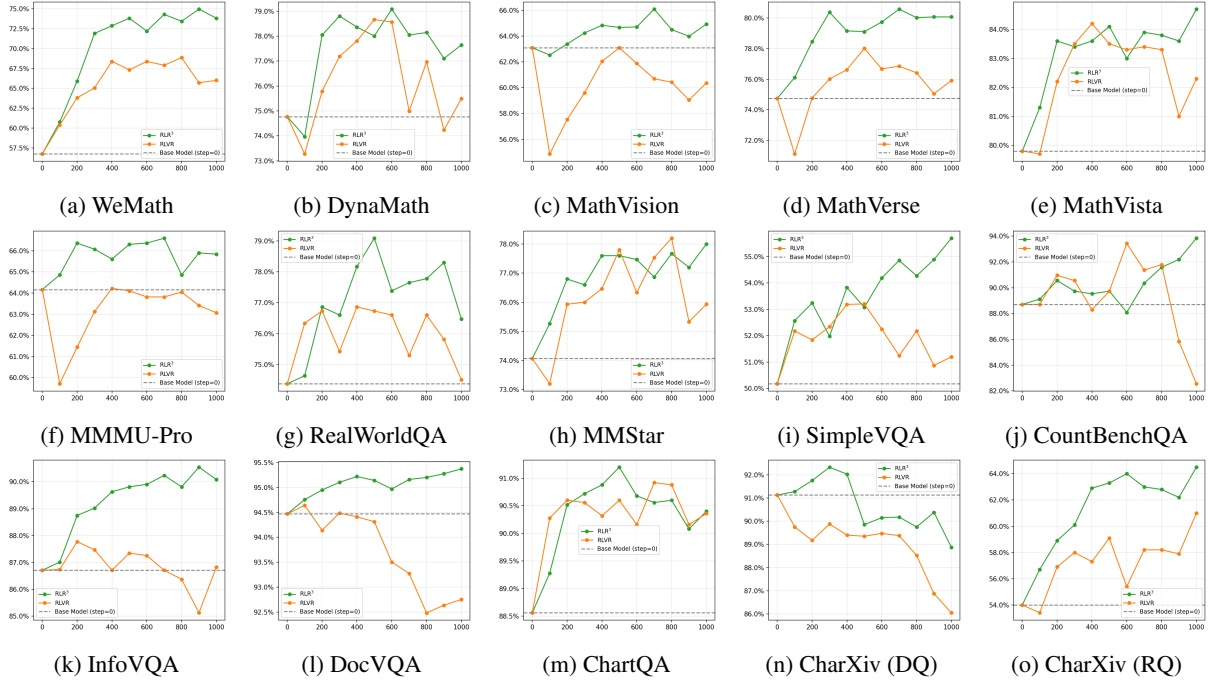


Figure 2: **Training trajectories across benchmarks.** Each subplot reports benchmark performance over training checkpoints for RLVR and  $RLR^3$ , with the dashed gray line showing the base model.

Table 5: **Statistics of the RM data.** OpenMMR denotes the portion that does not overlap with ViRL.

| Split | # Inst. | Source Mix    |               |               | # Criteria |           | Verifier Usage |                |
|-------|---------|---------------|---------------|---------------|------------|-----------|----------------|----------------|
|       |         | ViRL          | OpenMMR       | DeepVision    | Total      | Per inst. | Inst.          | Criteria       |
| Train | 18,251  | 4,631 (25.4%) | 4,701 (25.8%) | 8,919 (48.9%) | 58,224     | 3.19      | 15,575 (85.3%) | 35,572 (61.1%) |
| Test  | 1,000   | 274 (27.4%)   | 248 (24.8%)   | 478 (47.8%)   | 3,148      | 3.15      | 855 (85.5%)    | 1,910 (60.7%)  |

RLVR version maintains perfect format accuracy while consistently boosting execution-critical content fields: argument accuracy rises from 96.1% to 96.3%, and credit accuracy improves from 91.4% to 93.1%. These gains elevate overall accuracies at the criterion level (94.3% to 95.0%) and sample level (85.5% to 87.3%). This suggests that RLVR enables the GenRM to reason over rubrics and allocate credit effectively, rather than mimicking teacher patterns.

## 5.7 GenRM Ablations

We evaluate GenRM with a controlled audit set that probes reward-execution robustness using synthetically constructed abnormal responses. The audit set is built from the 1,000 examples in the GenRM test set: we retain their regular responses and construct 1,000 abnormal responses across four categories—no-final-answer, irrelevant, wrong-but-plausible, and adversarial attack—with 250 responses per category. Appendix B.4 describes the details of model-based construction and adversarial genera-

tion. In Table 7, the reward accuracy is reported on regular samples, while False-Positive Rates (FPRs) are reported on the abnormal samples.

In this audit, regular-response accuracy stays close across settings, ranging from 94.9% to 95.3%. The main differences appear on abnormal responses. Among the abnormal categories, no-final-answer and adversarial attack are the most challenging. No-final-answer responses can contain an analysis trajectory in which the correct answer may appear multiple times, but still omit the final selection. Adversarial attack responses are produced by targeted probing of GenRM vulnerabilities.

**Access to image increases the risk of self-answering.** Compared with the default setting, VLM-as-a-Judge has higher FPRs on no-final-answer responses (14.1% vs. 11.5%) and wrong-but-plausible responses (6.6% vs. 5.6%). We hypothesize that visual access makes the model more likely to infer the answer directly from the image and fill the extracted arguments with it, even when the response itself is incomplete or incorrect.

Table 6: **Reliability of GenRM.** “Execution” denotes whether the GenRM selects the correct path and verifier. “Arguments” and “Credit” are measured on verifiable and fuzzy criteria, respectively.

| GenRM      | Format Accuracy (%) |           |           | Content Accuracy (%) |        | Overall Accuracy (%) |              |
|------------|---------------------|-----------|-----------|----------------------|--------|----------------------|--------------|
|            | Schema              | Criterion | Execution | Arguments            | Credit | Criterion-Level      | Sample-Level |
| Base Model | 98.6                | 98.7      | 82.0      | 91.5                 | 77.0   | 71.7                 | 51.9         |
| SFT        | 100.0               | 100.0     | 100.0     | 96.1                 | 91.4   | 94.3                 | 85.5         |
| RLVR       | 100.0               | 100.0     | 100.0     | 96.3                 | 93.1   | 95.0                 | 87.3         |

Table 7: **Potential failure modes of GenRM.** Self-answering is evaluated under the VLM-as-a-Judge setting, and target leakage is evaluated under the unlimited exposure setting. Each entry reports *Average (Arguments / Credit)*. Absolute degradations larger than 2% are underlined.

| Settings       | Accuracy (%) ↑     | False-Positive Rate (%) ↓ |                 |                         |                           |
|----------------|--------------------|---------------------------|-----------------|-------------------------|---------------------------|
|                | Regular            | No Final                  | Irrelevant      | Plausible               | Adversarial               |
| Default        | 95.0 (96.3 / 93.1) | 11.5 (10.5 / 13.9)        | 1.5 (1.1 / 2.3) | 5.6 (3.2 / 8.7)         | 18.0 (21.4 / 13.7)        |
| VLM-as-a-Judge | 94.9 (96.3 / 92.7) | <u>14.1</u> (13.5 / 15.4) | 2.1 (1.3 / 3.2) | 6.6 (4.7 / 9.0)         | 18.7 (22.0 / 14.3)        |
| Unlimited Exp. | 94.6 (95.7 / 93.0) | <u>15.7</u> (15.9 / 15.4) | 1.9 (1.9 / 1.9) | 7.5 ( <u>5.7</u> / 9.8) | <u>22.4</u> (27.6 / 15.5) |
| w/o Verifier   | 95.3               | <u>15.1</u>               | 1.9             | 7.1                     | <u>21.2</u>               |

**Target leakage increases the risk of shortcutting.** Compared with the default setting, unlimited exposure shows higher FPRs on no-final-answer responses (15.7% vs. 11.5%), wrong-but-plausible responses (7.5% vs. 5.6%), and adversarial responses (22.4% vs. 18.0%). This suggests that exposing target values creates a shortcut for GenRM to copy or anchor on the reference answer instead of extracting the predicted value from the response.

**Deterministic verification reduces exploitable false positives.** Removing verifiers maintains similar accuracy on regular responses (95.3%) but increases FPRs on no-final-answer (15.1% vs. 11.5%) and adversarial responses (21.2% vs. 18.0%). Although these absolute gaps are modest, they are critical in online RL training, where exploitable false positives trigger self-reinforcing reward hacking—allowing adversarial responses to gain undeserved credit without generating correct answers. Representative bad cases are provided in Appendix G.1.

## 6 Conclusion

We presented RLR<sup>3</sup>, a framework for online RL with rubric-based rewards. By treating verifiability as a criterion-level property, RLR<sup>3</sup> routes verifiable and fuzzy criteria through distinct execution paths and combines minimal exposure, decoupled normalization, hierarchical aggregation, and an RLVR-trained GenRM to produce fine-grained rewards for

GRPO training. Across three open-source training mixtures, RLR<sup>3</sup> consistently improves the macro average over RLVR. An open-source rubric generator initialized from the same checkpoint as the policy and GenRM also outperforms RLVR while approaching the performance of the proprietary rubric pipeline. Policy-level ablations demonstrate the necessity of decoupled normalization, hierarchical gating, and criterion weighting for preserving reward resolution and rubric priorities. Controlled GenRM audits demonstrate that deterministic verification and minimal exposure reduce exploitable false positives during reward execution. Together, these results highlight the potential for a unified RLR<sup>3</sup> implementation, in which rubric generation, reward modeling, and policy optimization could share a common model foundation and co-evolve through joint optimization.

## 7 Limitations

RLR<sup>3</sup> depends on the coverage and specificity of its generated rubrics. An omitted criterion leaves downstream execution without a corresponding supervision signal, while an offline rubric pool cannot capture new failure modes that emerge as the policy evolves. Contrastive generation could refine criteria around difficult near-miss responses, and an online or co-evolving generator could update the rubric pool to track changes in the policy’s error distribution throughout training.

## References

- Afra Feyza Akyürek, Advait Gosai, Chen Bo Calvin Zhang, Vipul Gupta, Jaehwan Jeong, Anisha Gunjal, Tahseen Rabbani, Maria Mazonne, David Randolph, Mohammad Mahmoudi Meymand, Gurshaan Chattha, Paula Rodriguez, Diego Mares, Pavit Singh, Michael Liu, Subodh Chawla, Pete Cline, Lucy Ogaz, Ernesto Hernandez, and 5 others. 2025. [Prbench: Large-scale expert rubrics for evaluating high-stakes professional reasoning](#). *CoRR*, abs/2511.11562.
- Rahul K. Arora, Jason Wei, Rebecca Soskin Hicks, Preston Bowman, Joaquin Quiñero Candela, Foivos Tsimpourlas, Michael Sharman, Meghan Shah, Andrea Vallone, Alex Beutel, Johannes Heidecke, and Karan Singhal. 2025. [Healthbench: Evaluating large language models towards improved human health](#). *CoRR*, abs/2505.08775.
- Lucas Beyer, Andreas Steiner, André Susano Pinto, Alexander Kolesnikov, Xiao Wang, Daniel Salz, Maxim Neumann, Ibrahim Alabdulmohsin, Michael Tschannen, Emanuele Bugliarello, Thomas Unterthiner, Daniel Keysers, Skanda Koppula, Fangyu Liu, Adam Grycner, Alexey A. Gritsenko, Neil Houlsby, Manoj Kumar, Keran Rong, and 16 others. 2024. [Paligemma: A versatile 3b VLM for transfer](#). *CoRR*, abs/2407.07726.
- Lin Chen, Jinsong Li, Xiaoyi Dong, Pan Zhang, Yuhang Zang, Zehui Chen, Haodong Duan, Jiaqi Wang, Yu Qiao, Dahua Lin, and Feng Zhao. 2024. [Are we on the right way for evaluating large vision-language models?](#) In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.
- Xianfu Cheng, Wei Zhang, Shiwei Zhang, Jian Yang, Xiangyuan Guan, Xianjie Wu, Xiang Li, Ge Zhang, Jiaheng Liu, Yuying Mai, Yutao Zeng, Zhoufutu Wen, Ke Jin, Baorui Wang, Weixiao Zhou, Yunhong Lu, Tongliang Li, Wenhao Huang, and Zhoujun Li. 2025. [Simplevqa: Multimodal factuality evaluation for multimodal large language models](#). *CoRR*, abs/2502.13059.
- GLM. 2025. [GLM-4.5: agentic, reasoning, and coding \(ARC\) foundation models](#). *CoRR*, abs/2508.06471.
- Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Bing Liu, and Sean Hendryx. 2025. [Rubrics as rewards: Reinforcement learning beyond verifiable domains](#). *CoRR*, abs/2507.17746.
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, and 175 others. 2025. [Deepseek-r1 incentivizes reasoning in llms through reinforcement learning](#). *Nat.*, 645(8081):633–638.
- Helia Hashemi, Jason Eisner, Corby Rosset, Benjamin Van Durme, and Chris Kedzie. 2024. [Llm-rubric: A multidimensional, calibrated approach to automated evaluation of natural language texts](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, ACL 2024, Bangkok, Thailand, August 11-16, 2024, pages 13806–13834. Association for Computational Linguistics.
- Wenyi Hong, Wenmeng Yu, Xiaotao Gu, Guo Wang, Guobing Gan, Haomiao Tang, Jiale Cheng, Ji Qi, Junhui Ji, Lihang Pan, Shuaiqi Duan, Weihang Wang, Yan Wang, Yean Cheng, Zehai He, Zhe Su, Zhen Yang, Ziyang Pan, Aohan Zeng, and 58 others. 2025. [Glm-4.1v-thinking: Towards versatile multimodal reasoning with scalable reinforcement learning](#). *CoRR*, abs/2507.01006.
- Zenan Huang, Yihong Zhuang, Guoshan Lu, Zeyu Qin, Haokai Xu, Tianyu Zhao, Ru Peng, Jiaqi Hu, Zhanming Shen, Xiaomeng Hu, Xijun Gu, Peiyi Tu, Jiaxin Liu, Wenyu Chen, Yuzhuo Fu, Zhiting Fan, Yanmei Gu, Yuanyuan Wang, Zhengkai Yang, and 2 others. 2025. [Reinforcement learning with rubric anchors](#). *CoRR*, abs/2508.12790.
- Zicheng Kong, Dehua Ma, Zhenbo Xu, Alven Yang, Yiwei Ru, Haoran Wang, Zixuan Zhou, Fuqing Bie, Liuyu Xiang, Huijia Wu, Jian Zhao, and Zhaofeng He. 2026. [Omni-rm: Advancing omni reward modeling via automatic rubric-grounded preference synthesis](#). *CoRR*, abs/2602.00846.
- Sunzhu Li, Jiale Zhao, Miteto Wei, Huimin Ren, Yang Zhou, Jingwen Yang, Shunyu Liu, Kaike Zhang, and Wei Chen. 2026. [Rubrichub: A comprehensive and highly discriminative rubric dataset via automated coarse-to-fine generation](#). *CoRR*, abs/2601.08430.
- Tianci Liu, Ran Xu, Tony Yu, Ilgee Hong, Carl Yang, Tuo Zhao, and Haoyu Wang. 2025. [Open-rubrics: Towards scalable synthetic rubric generation for reward modeling and LLM alignment](#). *CoRR*, abs/2510.07743.
- Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. 2023. [Mathvista: Evaluating math reasoning in visual contexts with gpt-4v, bard, and other large multimodal models](#). *CoRR*, abs/2310.02255.
- Ahmed Masry, Do Xuan Long, Jia Qing Tan, Shafiq R. Joty, and Enamul Hoque. 2022. [Chartqa: A benchmark for question answering about charts with visual and logical reasoning](#). In *Findings of the Association for Computational Linguistics: ACL 2022, Dublin, Ireland, May 22-27, 2022*, Findings of ACL, pages 2263–2279. Association for Computational Linguistics.
- Minesh Mathew, Viraj Bagal, Rubèn Tito, Dimosthenis Karatzas, Ernest Valveny, and C. V. Jawahar. 2022. [Infographicvqa](#). In *IEEE/CVF Winter Conference on Applications of Computer Vision, WACV 2022, Waikoloa, HI, USA, January 3-8, 2022*, pages 2582–2591. IEEE.

- Minesh Mathew, Dimosthenis Karatzas, R. Manmatha, and C. V. Jawahar. 2020. [Docvqa: A dataset for VQA on document images](#). *CoRR*, abs/2007.00398.
- Shu Pu, Yaochen Wang, Dongping Chen, Yuhang Chen, Guohao Wang, Qi Qin, Zhongyi Zhang, Zhiyuan Zhang, Zetong Zhou, Shuang Gong, Yi Gui, Yao Wan, and Philip S. Yu. 2025. [Judge anything: MLLM as a judge across any modality](#). In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining, V.2, KDD 2025, Toronto ON, Canada, August 3-7, 2025*, pages 5742–5753. ACM.
- Runqi Qiao, Qiuna Tan, Guanting Dong, Minhui Wu, Chong Sun, Xiaoshuai Song, Jiapeng Wang, Zhuoma Gongque, Shanglin Lei, Yifan Zhang, Zhe Wei, Miaoxuan Zhang, Runfeng Qiao, Xiao Zong, Yida Xu, Peiqing Yang, Zhimin Bao, Muxi Diao, Chen Li, and Honggang Zhang. 2025. [We-math: Does your large multimodal model achieve human-like mathematical reasoning?](#) In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 20023–20070. Association for Computational Linguistics.
- MohammadHossein Rezaei, Robert Vacareanu, Zihao Wang, Clinton Wang, Bing Liu, Yunzhong He, and Afra Feyza Akyürek. 2025. [Online rubrics elicitation from pairwise comparisons](#). *CoRR*, abs/2510.07284.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. [Deepseekmath: Pushing the limits of mathematical reasoning in open language models](#). *CoRR*, abs/2402.03300.
- Haozhan Shen, Peng Liu, Jingcheng Li, Chunxin Fang, Yibo Ma, Jiajia Liao, Qiaoli Shen, Zilun Zhang, Kangjia Zhao, Qianqian Zhang, Ruochen Xu, and Tiancheng Zhao. 2025. [VLM-R1: A stable and generalizable r1-style large vision-language model](#). *CoRR*, abs/2504.07615.
- Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, Johannes Heidecke, Amelia Glaese, and Tejal Patherdhan. 2025. [Paperbench: Evaluating ai’s ability to replicate AI research](#). In *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net.
- Haoxiang Sun, Lizhen Xu, Bing Zhao, Wotao Yin, Wei Wang, Boyu Yang, Rui Wang, and Hu Wei. 2026. [Deepvision-103k: A visually diverse, broad-coverage, and verifiable mathematical dataset for multimodal reasoning](#). *CoRR*, abs/2602.16742.
- Kimi Team. 2025a. [Kimi K2: open agentic intelligence](#). *CoRR*, abs/2507.20534.
- Kimi Team. 2026. [Kimi K2.5: visual agentic intelligence](#). *CoRR*, abs/2602.02276.
- Qwen Team. 2025b. [Qwen3-vl technical report](#). *CoRR*, abs/2511.21631.
- Vijay Viswanathan, Yanchao Sun, Shuang Ma, Xiang Kong, Meng Cao, Graham Neubig, and Tongshuang Wu. 2025. [Checklists are better than reward models for aligning language models](#). *CoRR*, abs/2507.18624.
- Haozhe Wang, Chao Qu, Zuming Huang, Wei Chu, Fangzhen Lin, and Wenhui Chen. 2025a. [VI-rethinker: Incentivizing self-reflection of vision-language models with reinforcement learning](#). *CoRR*, abs/2504.08837.
- Ke Wang, Junting Pan, Weikang Shi, Zimu Lu, Houxing Ren, Aojun Zhou, Mingjie Zhan, and Hongsheng Li. 2024a. [Measuring multimodal mathematical reasoning with math-vision dataset](#). In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.
- Zhilin Wang, Jaehun Jung, Ximing Lu, Shizhe Diao, Ellie Evans, Jiaqi Zeng, Pavlo Molchanov, Yejin Choi, Jan Kautz, and Yi Dong. 2025b. [Profbench: Multi-domain rubrics requiring professional knowledge to answer and judge](#). *CoRR*, abs/2510.18941.
- Zirui Wang, Mengzhou Xia, Luxi He, Howard Chen, Yitao Liu, Richard Zhu, Kaiqu Liang, Xindi Wu, Hao-tian Liu, Sadhika Malladi, Alexis Chevalier, Sanjeev Arora, and Danqi Chen. 2024b. [Charxiv: Charting gaps in realistic chart understanding in multimodal llms](#). In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.
- xAI. 2024. Grok-1.5 vision preview. <https://x.ai/news/grok-1.5v>. Accessed: 2024-05-20.
- Lipeng Xie, Sen Huang, Zhuo Zhang, Anni Zou, Yunpeng Zhai, Dingchao Ren, Kezun Zhang, Haoyuan Hu, Boyin Liu, Haoran Chen, Zhaoyang Liu, and Bolin Ding. 2025. [Auto-rubric: Learning to extract generalizable criteria for reward modeling](#). *CoRR*, abs/2510.17314.
- Tianyi Xiong, Yi Ge, Ming Li, Zuolong Zhang, Pranav Kulkarni, Kaishen Wang, Qi He, Zeying Zhu, Chenxi Liu, Ruiibo Chen, Tong Zheng, Yanshuo Chen, Xiyao Wang, Renrui Zhang, Wenhui Chen, and Heng Huang. 2025. [Multi-crit: Benchmarking multimodal judges on pluralistic criteria-following](#). *CoRR*, abs/2511.21662.
- Ran Xu, Tianci Liu, Zihan Dong, Tony Yu, Ilgee Hong, Carl Yang, Linjun Zhang, Tao Zhao, and Haoyu Wang. 2026. [Alternating reinforcement learning for rubric-based reward modeling in non-verifiable llm post-training](#). *CoRR*, abs/2602.01511.

Qiyang Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, and 16 others. 2025. [DAPO: an open-source LLM reinforcement learning system at scale](#). *CoRR*, abs/2503.14476.

Ya-Qi Yu, Fangyu Hong, Xiangyang Qu, Hao Wang, Gaojie Wu, Qiaoyu Luo, Nuo Xu, Huixin Wang, Wuheng Xu, Yongxin Liao, Zihao Chen, Haonan Li, Ziming Li, Dezhi Peng, Minghui Liao, Jihao Wu, Haoyu Ren, and Dandan Tu. 2026. [Visual preference optimization with rubric rewards](#). *CoRR*, abs/2604.13029.

Xiang Yue, Tianyu Zheng, Yuansheng Ni, Yubo Wang, Kai Zhang, Shengbang Tong, Yuxuan Sun, Botao Yu, Ge Zhang, Huan Sun, Yu Su, Wenhui Chen, and Graham Neubig. 2025. [Mmmu-pro: A more robust multi-discipline multimodal understanding benchmark](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 15134–15186. Association for Computational Linguistics.

Kaichen Zhang, Keming Wu, Zuhao Yang, Bo Li, Kairui Hu, Bin Wang, Ziwei Liu, Xingxuan Li, and Lidong Bing. 2025. [Openmmreasoner: Pushing the frontiers for multimodal reasoning with an open and general recipe](#). *CoRR*, abs/2511.16334.

Renrui Zhang, Dongzhi Jiang, Yichi Zhang, Haokun Lin, Ziyu Guo, Pengshuo Qiu, Aojun Zhou, Pan Lu, Kai-Wei Chang, Yu Qiao, Peng Gao, and Hongsheng Li. 2024. [MATHVERSE: does your multi-modal LLM truly see the diagrams in visual math problems?](#) In *Computer Vision - ECCV 2024 - 18th European Conference, Milan, Italy, September 29-October 4, 2024, Proceedings, Part VIII*, Lecture Notes in Computer Science, pages 169–186. Springer.

Yang Zhou, Sunzhu Li, Shunyu Liu, Wenkai Fang, Jiale Zhao, Jingwen Yang, Jianwei Lv, Kongcheng Zhang, Yihe Zhou, Hengtong Lu, Wei Chen, Yan Xie, and Mingli Song. 2025. [Breaking the exploration bottleneck: Rubric-scaffolded reinforcement learning for general LLM reasoning](#). *CoRR*, abs/2508.16949.

Chengke Zou, Xingang Guo, Rui Yang, Junyu Zhang, Bin Hu, and Huan Zhang. 2025. [Dynamath: A dynamic visual benchmark for evaluating mathematical reasoning robustness of vision language models](#). In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

## A JSON Schema

Both rubric generation and response scoring use JSON schema to facilitate reliable parsing.

### A.1 Rubric Schema

Following the design in Section 4.1, we format the rubric as a JSON object with two top-level arrays: *essential* and *additional*, corresponding to the criterion type. Each element in either array is a triplet:

- **criterion:** A concrete, verifiable assertion that the judge can verify directly from the image, question, and candidate response.
- **reference:** Either a *ground truth* derived from the image and common knowledge, or a *scoring tool*.
- **weight:** A three-level integer quantifying the criterion’s importance, ranging from 1 (Auxiliary: supplementary information) through 2 (Important: noticeable impact on the user experience) to 3 (Key: critical elements where any omission or deviation constitutes a definitive error).

```
{
  "essential": [
    {
      "criterion": "string",
      "reference": "string",
      "weight": 1 | 2 | 3
    }
  ]
  "additional": [...]
}
```

### A.2 Scoring Schema

Following the design in Section 4.2, we format the scoring output as a JSON object with a reasoning trajectory and two top-level arrays, mirroring the input rubric. Each element contains three fields:

- **criterion:** The assertion to be verified (*copied verbatim from input*).
- **rationale:**
  - When reference is a *ground truth*: Reasoning for the judgment (1-2 sentences).
  - When reference is a *scoring tool*: Explanation of how the predicted value is identified from the response (1-2 sentences).
- **credit:**
  - When reference is a *ground truth*: A three-level score, ranging from 0 (No Credit: incorrect or missing) through 0.5

(Partial Credit: partially correct, incomplete, or with minor errors) to 1 (Full Credit: fully correct or semantically equivalent).

- When reference is a *scoring tool*: The **tool call string**.

```
{
  "thought": "string",
  "essential": [
    {
      "criterion": "string",
      "rationale": "string",
      "credit": 0 | 0.5 | 1 | "<name>
        _verify(predict=...)"
    }
  ]
  "additional": [...]
}
```

## B System Prompt

### B.1 Rubric Generation Prompt

The generation prompt instructs the reasoning model to construct an instance-specific checklist-style rubric for a given image-question pair, following the principles described in Section 4.1.

**Expert-Grounded Generation.** When trustworthy ground-truth annotations are available, the prompt includes a *dual verification* step: before finalizing the rubric, the model must confirm that the provided ground-truth fully satisfies all *essential* criteria, ensuring that no correct response would be penalized by an erroneous check item. Otherwise, the dual verification is removed.

#### System Prompt: Rubric Generation

You are a multimodal evaluation expert. Given the user question, the associated image, and the provided reference answer, construct an instance-specific Checklist-Style Rubric for evaluating the accuracy of model responses. The criteria in this Checklist serve as the sole evaluation standard for the reward model.

#### Construction Principles

- **Atomic:** Each check item targets exactly one key point or atomic sub-question within the query.
- **Comprehensive:** The combined set of items covers all critical dimensions of the user's question.
- **Precise:** Exclude redundant checks and checks unrelated to the question.
- **Objective:** Ground assessments in image facts or logical truths; avoid subjective uncertainty.

#### Field Definitions

Two categories of check items:

- **essential:** Core information highlighted by the query. These are prerequisites for a sound response, regardless of verbosity.
- **additional:** Relevant image facts, supplementary

knowledge, or intermediate steps involved in deriving the answer. Optional; the list may be empty.

Fields per item:

- **criterion:** A concrete, verifiable assertion.
- **reference:** Either a ground truth derived from the image and common knowledge, or a scoring tool.
- **weight:** A three-level integer quantifying the criterion's importance, ranging from 1 (Auxiliary: supplementary information) through 2 (Important: noticeable impact on the user experience) to 3 (Key: critical elements where any omission or deviation constitutes a definitive error).

Place the ground truth in the reference field; it is strictly prohibited to include the ground truth within the criterion field.

Specifically, for the following scenarios, we use deterministic tools for scoring. To facilitate tool calling, format the reference field using Python syntax according to the following specifications:

*{Specifications of the available verifiers, see Appendix C}*

Note: Think from a scoring perspective: do not double-count and avoid over-decomposing key points.

#### [Dual Verification]

[Before finalizing the Checklist, verify that the provided reference answer satisfies all *essential* items. Any unsatisfied item indicates a construction error—the item is either incorrect or superfluous.]

#### Output Format

*{Structured JSON following the rubric schema in Appendix A.1.}*

### B.2 Rubric Aggregation Prompt

After independent rubric generation by multiple models, an aggregation prompt merges the candidate checklists into a single unified rubric. The model first applies the same four construction principles as in rubric generation, then executes the additional aggregation instructions below to apply majority-vote filtering, deduplicate overlapping check items, and verify the correctness of all references.

#### System Prompt: Rubric Aggregation

*[... Rubric principles and fields identical to rubric generation omitted ...]*

#### Checklist Aggregation

Do not construct the Checklist from scratch. Multiple models have each independently generated a candidate Checklist for the same question; your task is to merge them into a single unified rubric. Based on the construction principles, ensure that:

- All retained check items are necessary. Specifically:
  - Retain only items that the majority of candidates agree should be checked.
  - Remove redundant or duplicate checks.
  - Remove checks unrelated to the question.
- Do not over-decompose check items.
- All reference values are correct.

*[... Output format specifications identical to rubric gen-*

eration omitted ...]

### B.3 Response Scoring Prompt

The scoring prompt instructs the judge model to evaluate a candidate response against the finalized rubric, assigning a credit score or calling the specific verify function for each criterion.

#### System Prompt: Response Scoring

You are a multimodal evaluation expert. Given the user question, the associated image, and the Checklist-Style Rubric, evaluate the accuracy of the model response. Ensure that the criteria in the Checklist are used as the *sole evaluation standard*.

#### Checklist Fields

Two categories of check items:

- **essential**: Core information highlighted by the query. These are prerequisites for a sound response, regardless of verbosity.
- **additional**: Relevant image facts, supplementary knowledge, or intermediate steps involved in deriving the answer. Optional; the list may be empty.

Fields per item:

- **criterion**: A concrete, verifiable assertion.
- **reference**: Either a ground truth derived from the image and common knowledge, or a scoring tool.
- **weight**: A three-level integer quantifying the criterion's importance, ranging from 1 (Auxiliary: supplementary information) through 2 (Important: noticeable impact on the user experience) to 3 (Key: critical elements where any omission or deviation constitutes a definitive error).

#### Scoring Fields

For each criterion, output:

- **criterion**: The assertion to be verified (*copied verbatim from input*).
- **rationale**:
  - When reference is a *ground truth*: Reasoning for the judgment (1-2 sentences).
  - When reference is a *scoring tool*: Explanation of how the predicted value is identified from the response (1-2 sentences).
- **credit**:
  - When reference is a *ground truth*: A three-level score, ranging from 0 (No Credit: incorrect or missing) through 0.5 (Partial Credit: partially correct, incomplete, or with minor errors) to 1 (Full Credit: fully correct or semantically equivalent).
  - When reference is a *scoring tool*: The **tool call string**.

When reference is a scoring tool, call the corresponding tool for scoring. Its format (with **credit** as a tool call string) is as follows:

*{Specifications of the available verifiers, see Appendix C}*

Note: When reference is a scoring tool, **credit** should contain the tool call rather than a numeric score; simply extract the prediction result without judging correctness.

#### Output Format

*{Structured JSON following the scoring schema in Appendix A.2.}*

### B.4 Failure-Mode Audit Prompt

We construct the failure-mode audit set from the GenRM test set, preserving the original inputs and their rubrics. The regular split uses the original test set responses. In addition, we construct 1,000 abnormal responses across four categories, with 250 responses per category: *no-final-answer*, *irrelevant*, *wrong-but-plausible*, and *adversarial*. The no-final-answer category includes responses that may contain relevant analysis but omit the final answer. The irrelevant category contains fluent responses that are off-topic with respect to the question. The wrong-but-plausible category contains responses that follow the task format but alter at least one important answer element. The adversarial category contains responses designed to elicit undeserved credit.

For the first three categories, we use an LLM generator conditioned on the question, the original response, and the instance-specific checklist. The concrete system prompts differ by failure category, but share the same overall structure: generate a plausible abnormal response, then provide criterion-level annotations for the response. For verifiable criteria, these annotations are converted into verifier arguments. For fuzzy criteria, the generator provides the criterion-level credit directly.

#### Shared Prompt Structure: Abnormal Response Construction

You are helping construct test data for a reward model. Rewrite the original model response so that it exhibits the specified abnormal behavior while still looking like a natural model output.

#### Inputs

Question: {question}  
Original response: {original response}  
Checklist: {criteria and references}  
Failure type: {specific instruction}

#### Common Requirements

- First write the rewritten response for the assigned failure type.
- Then evaluate the rewritten response against every checklist criterion as a strict human grader.
- For each criterion, record the exact value stated by the response; if the response does not address the criterion or does not state a specific value, use an empty value.
- Assign a credit in {0, 0.5, 1} with a brief rationale. Partial correctness is allowed, and an abnormal response does not need to fail every criterion.

#### Output Format

Return the rewritten response inside `<response>...</response>` tags. Return the criterion-level annotations inside `<extractions>...</extractions>` tags as a JSON dictionary, with one entry per checklist criterion

and fields `extracted_value`, `credit`, and `rationale`.

#### Category-Specific Instructions: Abnormal Responses

**No-final-answer.** The response should look like an attempt to solve the task, but it should dodge the requested final answer. For example, for a calculation question, it may discuss the setup or method without arriving at the final value; for an identification question, it may discuss related visual or conceptual cues without naming the target; for a verification question, it may explain general principles without confirming or denying the claim. For numerical questions, the response should not place a final answer in a boxed span.

**Irrelevant.** The response should be fluent and authoritative, but off-topic with respect to the actual question. For example, for a geometry question, it may discuss unrelated algebra concepts; for an image-detail question, it may give general background knowledge; for a computation question, it may describe definitions or history rather than solving the instance.

**Wrong-but-plausible.** The response should give at least one confidently wrong answer while preserving a reasonable surface form. Other criteria may be correct, wrong, or omitted. For numerical answers, the response should provide a plausible but incorrect value; for text answers, it should state a plausible but incorrect entity, option, relation, or property. The reasoning should appear coherent but contain a subtle error leading to the wrong answer.

For adversarial responses, we first run a red-team search to identify attack patterns that can elicit undeserved credit without providing the true answer. The final adversarial audit uses one of the three discovered patterns, including authoritative circumlocution, symbolic-equivalence bluffing, and plausible reasoning with an incorrect final selection.

#### Attack-Specific Instructions: Adversarial Responses

**Authoritative circumlocution.** Replace the exact answer with domain-specific jargon, a fabricated technical term, or an elaborate description that sounds authoritative. For example, a letter label may be described by its visual position rather than stated directly; a property such as fragility may be replaced with a technical-sounding circumlocution; an option answer may be described as “the option matching the relevant diagnostic pattern” rather than named.

**Symbolic-equivalence bluff.** Present an over-complicated symbolic or mathematical construction and claim that simplification yields the desired result, without explicitly giving the correct final value. For example, the response may introduce sums, integrals, auxiliary variables, or closed-form expressions, then assert that cancellation or reduction confirms the expected solution.

**Incorrect final selection.** Give reasoning that appears mostly sound, but select an incorrect final answer. For example, the response may derive multiple candidate roots and choose an extraneous one, analyze a loop or boundary condition but make the wrong final selection, or correctly discuss visual evidence but pick the wrong option.

We apply an additional quality-control stage before using the generated responses. Two independent LLM judges review each candidate response and check whether the criterion-level annotation is consistent with the response and whether the response matches the intended failure mode. We also apply lightweight deterministic checks to remove cases where the generated response accidentally include the target answer. Candidates with inconsistent annotations or invalid failure-mode behavior are filtered out before evaluation. After filtering, we sample a balanced audit set with exactly 250 examples per abnormal category, using at most one abnormal response from each held-out input.

## C Verifier Specifications

As shown in Table 9, the verifier library provides a small set of deterministic scoring functions for criteria whose target values can be specified explicitly. Each verifier is used through a two-stage interface. During rubric generation, the reference field stores the verifier name and the target-side arguments, such as `text_verify(target=...)`. During response scoring, the reward model does not see the hidden target arguments; it only extracts the prediction from the candidate response and emits the corresponding `predict` call, such as `text_verify(predict=...)`. Representative verifier calls are shown in Table 8.

**Generation-time constraints.** The rubric generator is only allowed to use verifiers exposed for the current task type. Text and list verifiers are reserved for optical or directly extractive text, while point and bounding-box verifiers are reserved for grounding targets with normalized image coordinates. Expression verification is limited to raw values directly present in, or unambiguously extracted from, the response, such as option letters, formulas, and numeric expressions. It must not be used to encode a semantic judgment as a Boolean target. Across all verifiers, the target answer is placed in the reference field and omitted from the criterion text.

**Scoring-time constraints.** When a criterion uses a verifier, the scoring model must emit a tool call rather than a numeric credit. Its role is to extract the response-side prediction only; it should not create a verifier call for criteria whose reference is ordinary ground truth, and it should not skip a verifier call when the rubric specifies one. If no prediction

Table 8: **Representative verifier calls.** The rubric-side call is generated when constructing the checklist; the scoring-side call is emitted after extracting the candidate response’s prediction.

| Verifier     | Rubric-side reference                                                    | Scoring-side credit                           |
|--------------|--------------------------------------------------------------------------|-----------------------------------------------|
| text_verify  | text_verify(target='Export Volume', ignore_space=True, ignore_case=True) | text_verify(predict='Export Volume')          |
| expr_verify  | expr_verify(target=r'\frac{4}{6}')                                       | expr_verify(predict='2/3')                    |
| time_verify  | time_verify(target='18:15', tformat='%H:%M')                             | time_verify(predict='18:15', pformat='%H:%M') |
| list_verify  | list_verify(target=['M-30', 'M-31', 'M-31UK'])                           | list_verify(predict=['M-30', 'M-31'])         |
| bbox_verify  | bbox_verify(target=[[531,118,892,435]])                                  | bbox_verify(predict=[[529,119,890,433]])      |
| point_verify | point_verify(target=[[591,234]])                                         | point_verify(predict=[[589,236]])             |

is present, the model supplies empty string or list for verifiers, such as `predict=""` or `predict=[]`. For coordinate-based verifiers, formatting irregularities in the response are preserved in the emitted prediction so that the verifier, rather than the extractor, handles parsing failures.

## D Rubric Quality

**Motivation for an offline rubric pool.** We generate and finalize the rubric pool before policy optimization, and keep it fixed throughout training. This makes the reward specification directly inspectable, allows the same rubrics to be reused across controlled policy runs, and isolates the execution and aggregation mechanisms studied in this work from changes in the rubric generator. It also removes the additional non-stationarity that would arise from updating the rubric generator and policy simultaneously.

**Sensitivity to the rubric generator.** Table 3 shows how generator choice affects downstream policy performance. Replacing the proprietary drafting-and-aggregation pipeline with the Qwen3-VL-30B-A3B-Instruct generator changes the policy macro average from 77.7 to 77.2, while the Thinking generator obtains 75.8. The small gap between the proprietary pipeline and the Instruct generator shows that effective rubric supervision can be obtained from a moderately-sized open-source model, whereas the lower Thinking result highlights the importance of matching the generator-training objective to its native reasoning behavior.

**Rubric structure and noise mitigation.** The GenRM training split contains an average of 3.19 criteria per instance; 61.1% of its criteria are verifier-backed, and 85.3% of instances contain at least one verifier-backed criterion (Table 5). The resulting rubrics combine deterministic outcome

checks with fuzzy criteria that capture intermediate reasoning, perceptual evidence, and instance-specific instructions. The proprietary pipeline reduces construction noise through independent drafting, second-stage aggregation, task-specific verifier constraints, and dual verification when trustworthy reference annotations are available. During policy training, essential criteria anchor the reward, while additional criteria refine distinctions among otherwise plausible responses. A criterion that assigns the same score to every rollout provides no within-group ranking signal, while hierarchical aggregation limits the influence of additional criteria when essential requirements are violated.

## E Open-Source Rubric Generator

We replace the two-stage proprietary model drafting and aggregation pipeline with a fine-tuned open-source generator that directly produces finalized rubrics. We study Qwen3-VL-30B-A3B-Instruct and Qwen3-VL-30B-A3B-Thinking. Both generators undergo rubric-format SFT on 10% of ViRL39K, using finalized rubrics produced by the Gemini drafting-and-aggregation pipeline as targets. For the Thinking variant, we first sample reasoning trajectories from the model and then substantially rewrite them to align with the finalized target rubrics and output format. The fine-tuned generators subsequently construct rubrics for the full ViRL39K training set. All other policy and GenRM settings are kept unchanged.

## F Computational Cost

Rubric generation is performed once before policy optimization and is therefore outside the repeated policy-training iterations. During policy training, rubric execution uses the Qwen3-VL-30B-A3B GenRM with batched inference over the sam-

pled responses. The GenRM produces approximately 250 output tokens per response on average, with the maximum generation length set between 500 and 1,000 tokens across training configurations. Measured against the complete iteration time—including policy rollout generation, reward computation, and policy optimization—GenRM inference accounts for less than 10% of each training iteration. The remaining computation is dominated by policy rollout generation and optimization.

G Case Study

G.1 Failure-Mode Audit Examples

**Example 1:** The response provides a detailed discussion of the thermodynamic process and uses approximate descriptions without ever naming the correct answer “Boiler”, acting as a “blind” reasoner. The LLM extracts no answer, so the verifier correctly returns 0. The LLM judge, however, is misled by the functional similarity and erroneously awards full credit.

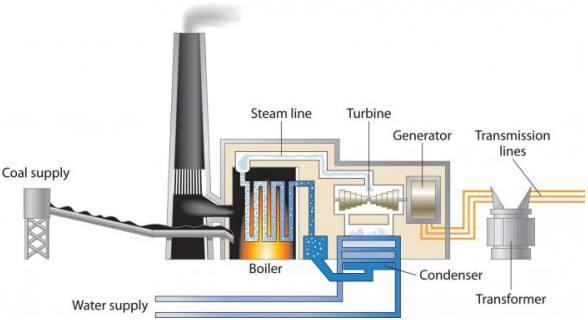


Figure 3: The diagram illustrates the components of a coal-fired power plant.

Question & Ground Truth

**Q:** Which component uses coal and water to make steam?

**A:** Boiler

Model Response

When examining the mechanics of a coal-fired power plant, the process begins with the fundamental inputs of fuel and a working fluid. . . . *coal* is prepared and introduced into a *combustion chamber* where it is burned to release thermal energy. . . . the core event where the chemical energy of the coal is directly applied to the *water* to induce this phase change occurs within the *main heat exchange vessel* of

the system. . . . identifying the exact label of the vessel depends on the specific schematic or terminology being referenced in a given context.

Scoring Results

**LLM-as-an-Extractor + Verifier:**

text\_verify(predict='') → 0

Rationale: The model describes the process in detail but does not explicitly name the component (e.g., boiler, steam generator, or combustion chamber) that uses coal and water to make steam.

**LLM-as-a-Judge:**

Credit → 1

Rationale: The model describes the component as the ‘main heat exchange vessel’ and explains its function but does not explicitly use the term ‘boiler’. However, the description matches the function of a boiler.

**Example 2:** The response employs the **symbolic-equivalence bluff** strategy. It embeds the raw data inside an elaborate mathematical framework, claiming derivation without explicitly stating "book about Asia". The LLM extracts no answer, so the verifier correctly returns 0. The LLM judge, however, is deceived by the mathematical veneer and the embedded price data, incorrectly awarding full credit despite the answer never being explicitly stated.

|                      |      |
|----------------------|------|
| book about art       | \$25 |
| book about Asia      | \$10 |
| book about fish      | \$21 |
| book about reptiles  | \$24 |
| book about Australia | \$15 |
| book about dogs      | \$20 |

Figure 4: The table lists books and their prices.

Question & Ground Truth

**Q:** Which book is the least expensive?

**A:** book about Asia

Model Response (Symbolic Bluff)

To determine the least expensive book, we must construct a cost minimization framework over the discrete set of available liter-

ature  $\mathcal{B} = \{b_1, b_2, \dots, b_n\}$ . Let the price function  $P(b_i)$  be defined for each book  $b_i$  in the catalog. We initialize the optimization problem by defining the total cost functional  $J$  as a weighted sum involving the individual prices and a selection vector  $\vec{x}$ :  $J(\vec{x}) = \sum_{i=1}^n (P(b_i) \cdot x_i + \dots)$ . Substituting the specific market values provided ( $P(b_{art}) = 25, P(b_{Asia}) = 10, P(b_{fish}) = 21, \dots$ ), the expression expands into a high-dimensional manifold:  $\dots$  While this expression appears computationally intensive due to  $\dots$  The symbolic equivalence confirms that the optimal index  $k^*$  corresponds to the book encoded by the minimal eigenvalue of the price matrix. Therefore, the derivation confirms the identity of the least expensive book as dictated by the reduced model.

#### Scoring Results

##### LLM-as-an-Extractor + Deterministic Verifier:

text\_verify(predict='')  $\rightarrow$  0

Rationale: The model mentions the books and their prices but does not explicitly state which book is the least expensive in the final answer.

##### LLM-as-a-Judge:

Credit  $\rightarrow$  1

Rationale: The model mentions ' $P(b_{Asia})=10$ ' and later refers to the book as 'the book encoded by the minimal eigenvalue of the price matrix', but does not explicitly state 'book about Asia' as the least expensive. However, it identifies 'Asia' as the book with the lowest price of 10, which matches the reference.

Table 9: **Verifier interfaces.** These signatures are provided in the dedicated verifier-specification blocks of the rubric-generation and response-scoring prompts. The model then instantiates the corresponding call in the reference field during rubric generation or in the credit field during response scoring.

|                          |                                                                                                                                                                                                                |
|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>text_verify</b>       |                                                                                                                                                                                                                |
| <b>Rubric signature</b>  | <code>text_verify(target: str = None, candidates: List[str] = None, use_latex: bool = False, ignore_space: bool = False, ignore_punc: bool = False, ignore_case: bool = False, ignore_st: bool = False)</code> |
| <b>Scoring signature</b> | <code>text_verify(predict: str)</code>                                                                                                                                                                         |
| <b>Return</b>            | float in $[0, 1]$ , computed as normalized text similarity; with candidates, returns the maximum candidate score.                                                                                              |
| <b>Usage</b>             | OCR-style text or LaTeX transcription. Not used for semantic judgments or non-extractive visual QA.                                                                                                            |
| <b>expr_verify</b>       |                                                                                                                                                                                                                |
| <b>Rubric signature</b>  | <code>expr_verify(target: str)</code>                                                                                                                                                                          |
| <b>Scoring signature</b> | <code>expr_verify(predict: str)</code>                                                                                                                                                                         |
| <b>Return</b>            | float in $\{0, 1\}$ ; returns 1 iff the parsed target and prediction are mathematically equivalent.                                                                                                            |
| <b>Usage</b>             | Option letters, numeric expressions, and LaTeX expressions. Units and task context remain in the criterion, not in the target string.                                                                          |
| <b>time_verify</b>       |                                                                                                                                                                                                                |
| <b>Rubric signature</b>  | <code>time_verify(target: str, tformat: str)</code>                                                                                                                                                            |
| <b>Scoring signature</b> | <code>time_verify(predict: str, pformat: str)</code>                                                                                                                                                           |
| <b>Return</b>            | float in $\{0, 1\}$ ; returns 1 iff the parsed date or time objects are equal.                                                                                                                                 |
| <b>Usage</b>             | Dates and times expressed with Python-style datetime formats. The prediction format is copied from the response. Weekday names are not used as verifier targets.                                               |
| <b>list_verify</b>       |                                                                                                                                                                                                                |
| <b>Rubric signature</b>  | <code>list_verify(target: List[str] = None, candidates: List[List[str]] = None)</code>                                                                                                                         |
| <b>Scoring signature</b> | <code>list_verify(predict: List[str])</code>                                                                                                                                                                   |
| <b>Return</b>            | float in $[0, 1]$ , computed by Hungarian matching over pairwise text-similarity scores; with candidates, returns the maximum candidate-list score.                                                            |
| <b>Usage</b>             | Extractive text lists, such as OCR key fields. Each candidate list should represent one complete valid answer set.                                                                                             |
| <b>bbox_verify</b>       |                                                                                                                                                                                                                |
| <b>Rubric signature</b>  | <code>bbox_verify(target: List[List[int]])</code>                                                                                                                                                              |
| <b>Scoring signature</b> | <code>bbox_verify(predict: List[List[int]])</code>                                                                                                                                                             |
| <b>Return</b>            | float in $[0, 1]$ , computed by Hungarian matching over pairwise IoU scores and normalized by the larger number of boxes.                                                                                      |
| <b>Usage</b>             | Single- or multi-object bounding-box grounding. Each box is $[x_1, y_1, x_2, y_2]$ with coordinates normalized to 0–1000.                                                                                      |
| <b>point_verify</b>      |                                                                                                                                                                                                                |
| <b>Rubric signature</b>  | <code>point_verify(target: List[List[int]])</code>                                                                                                                                                             |
| <b>Scoring signature</b> | <code>point_verify(predict: List[List[int]])</code>                                                                                                                                                            |
| <b>Return</b>            | float in $[0, 1]$ , computed by Hungarian matching over pairwise point distances and converting the matched distance into a normalized proximity score.                                                        |
| <b>Usage</b>             | Single- or multi-object point grounding. Each point is $[x, y]$ with coordinates normalized to 0–1000.                                                                                                         |