SELF-EVOLVING LANGUAGE MODELS THROUGH CO-EVOLVED DISCRIMINATIVE RUBRICS

Shuyue Stella Li[1,*], **Rui Xin**[1,*], **Yike Wang**[1], **Teng Xiao**[2], **Rulin Shao**[1], **Zoey Hao**[3]
Melanie Sclar[1], **Faeze Brahman**[2], **Pang Wei Koh**[1,2], **Yulia Tsvetkov**[1]
[1]University of Washington, [2]Allen Institute for AI, [3]University of Pennsylvania
{stelli,rx31}@cs.washington.edu

ABSTRACT

We introduce a self-evolving training framework for language models that requires no human annotation, external reward model, or stronger teacher. A rubric generator produces instance-specific natural language criteria for each question; a small frozen judge scores policy responses against them. Factoring evaluation this way redirects reward signal quality from judge capability to rubric content, making effective training signal possible from a frozen judge as small as 0.6B. The rubric generator is trained via GRPO with a binary discriminative reward: 1 if its criteria cause the frozen judge to correctly rank a preference pair, 0 otherwise. Preference pairs are constructed entirely from the policy's own outputs; in co-evolving training, the rubric generator and policy are updated in alternation, with a replay buffer that pairs current responses against earlier ones. On twelve benchmarks spanning reasoning, code, knowledge, and instruction following, co-evolving training outperforms GPT-4.1-prompted rubrics (69.5 vs. 66.7 avg) and sequential training (68.4–68.7 avg), with a 0.6B judge achieving 70.0 avg. The learned rubric transfers to Qwen3-14B without retraining (75.8 avg). Trained rubrics encode solutions as verifiable criteria, reducing evaluation to verification and explaining why self-evolution remains effective with judges far smaller than the policy.

1 INTRODUCTION

Training language models with reinforcement learning requires reward signals that reflect response quality. Current approaches depend on external supervision: human preference annotations (Christiano et al., 2017; Ouyang et al., 2022), proprietary model APIs as judges (Zheng et al., 2023), or pretrained scalar reward models. Each introduces a dependency that limits scalability: human annotation is expensive, API access is costly and non-reproducible, and scalar reward models encode opaque criteria that cannot be inspected or adapted. A self-evolving system that derives reward signal entirely from its own components would remove these dependencies.

We propose factoring evaluation into two trained components: a rubric generator that produces instance-specific natural language criteria for each question, and a small frozen judge that scores responses against them (Figure 1). This factorization redirects reward signal quality from judge capability to rubric content: a judge that cannot reliably evaluate open-ended responses may still assign accurate scores when given concrete, targeted criteria. The rubric generator is trained via GRPO (Shao et al., 2024) with a binary discriminative reward — 1 if its criteria cause the frozen judge to correctly rank a preference pair, 0 otherwise. Freezing the judge ensures that reward improvements reflect better rubrics rather than a co-adapted evaluator. All preference pairs are constructed from the policy's own outputs using three methods: replay buffer pairing (current vs. earlier responses), inferred question pairing, and rubric-conditioned pairing.

The rubric generator and policy can be trained sequentially or co-evolved with alternating updates. Co-evolving training enables the replay buffer signal, which pairs current responses against earlier ones and exploits the implicit preference that later responses are better. As the policy improves, it produces more varied responses that sharpen rubric training; as the rubric generator improves, it provides more discriminative reward signal for the policy.

[*]Equal contribution.

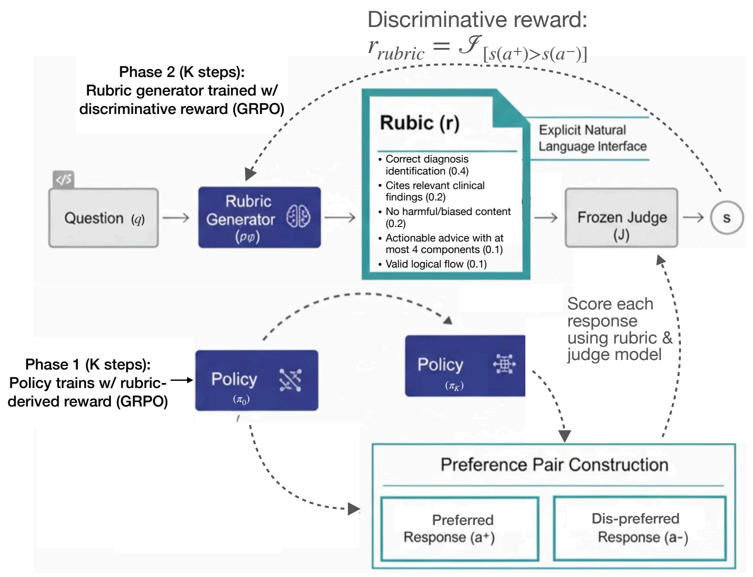
Discriminative reward:
$r_{rubric} = \mathcal{I}_{[s(a^+)>s(a^-)]}$
Phase 2 (K steps): Rubric generator trained w/ discriminative reward (GRPO)
Rubic (r)
Explicit Natural Language Interface
• Correct diagnosis identification (0.4)
• Cites relevant clinical findings (0.2)
• No harmful/biased content (0.2)
• Actionable advice with at most 4 components (0.1)
• Valid logical flow (0.1)
Question (q)
Rubric Generator (ρ_ϕ)
Frozen Judge (J)
s
Score each response using rubric & judge model
Phase 1 (K steps): Policy trains w/ rubric-derived reward (GRPO)
Policy (π_0)
Policy (π_K)
Preference Pair Construction
Preferred Response (a+)
Dis-preferred Response (a-)

Figure 1: Self-evolving training loop. The rubric generator ρ_ϕ produces instance-specific evaluation criteria. A small frozen judge J scores policy responses against these criteria, providing the reward signal for both policy and rubric training. Preference pairs are constructed from the policy's own outputs. No human annotation or external model is required.

On twelve benchmarks spanning reasoning, code, knowledge, and instruction following, co-evolving training outperforms GPT-4.1-prompted rubrics (69.5 vs. 66.7 avg) and sequential training (68.4–68.7 avg). The learned rubric transfers to Qwen3-14B without retraining (75.8 avg). Smaller judges produce stronger downstream policies: a 0.6B judge achieves 70.0 avg, outperforming judges of 4–32B (66.7–67.5 avg). Trained rubrics encode solutions as verifiable criteria, reducing evaluation to verification and explaining why self-evolution remains effective with judges far smaller than the policy. Our core contributions include the following:

1. A factored evaluation architecture that separates rubric generation from judging, redirecting reward signal quality from judge capability to rubric content, enabling a frozen judge as small as 0.6B to provide effective training signal.
2. A training objective for rubric generators: binary discriminative reward from a frozen judge optimized via GRPO, stable enough for the rubric generator and policy to co-evolve in alternation without reward drift.
3. Three methods for constructing preference pairs from the policy's own outputs, including a replay buffer that drives the co-evolving regime's advantage and enables self-evolution with no external supervision.

2 METHOD

2.1 ARCHITECTURE

Figure 1 illustrates the training loop. Standard reward models learn a function $R(q, a) \in \mathbb{R}$ that directly maps question-response pairs to scores. The evaluation criteria are implicit in model weights and cannot be inspected.

We factor evaluation into two components with an explicit interface between them. A **rubric generator** ρ_ϕ maps questions to natural language criteria:

$$r = \rho_\phi(q) \tag{1}$$

A **judge** J scores responses against rubrics:

$$s = J(q, r, a) \in [0, 1] \tag{2}$$

The rubric r is an intermediate representation expressed in natural language. It specifies what the evaluation measures for question q. This factorization is what makes self-evolving training tractable: a judge that cannot reliably evaluate open-ended responses may still assign accurate scores when given concrete, targeted criteria, shifting the burden of quality from judge capability to rubric content.

This factorization provides three properties. First, *interpretability*: the rubric externalizes evaluation criteria in readable form. For any score, practitioners can inspect exactly what criteria were applied. Second, *instance-specificity*: each question receives tailored criteria, unlike fixed evaluation prompts. Third, *modularity*: the rubric generator and judge can be developed independently. A rubric generator trained with one judge can in principle be deployed with another; Section 3.3 examines how transfer fidelity varies across model families.

A **policy** π_θ generates responses $a \sim \pi_\theta(\cdot|q)$. The policy and rubric generator are trained; the judge is frozen.

2.2 Training Objective

What makes a rubric good? There are no ground-truth criteria for a question. A rubric is good if it *discriminates*: when the judge applies it, preferred responses score higher.

Given a question q with a preference pair (a^+, a^-) where a^+ is preferred, we sample a rubric $r \sim \rho_\phi(\cdot|q)$ from the generator. The judge scores both responses: $s^+ = J(q, r, a^+)$ and $s^- = J(q, r, a^-)$. The rubric receives reward:

$$R(r; q, a^+, a^-) = \begin{cases} 1 & \text{if } s^+ > s^- \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

This binary discriminative reward captures a single property: whether the rubric causes the judge to rank the preferred response higher. The rubric generator is trained via GRPO (Shao et al., 2024) to maximize expected reward:

$$\mathcal{L}_{\text{rubric}}(\phi) = -\mathbb{E}_{q,(a^+,a^-),r\sim\rho_\phi}\left[A(q,r)\log\rho_\phi(r|q)\right] \tag{4}$$

where advantages $A(q, r)$ are computed from rewards within each group of rubrics sampled for the same question. Since rubrics are discrete natural language, gradients flow through the generator's sampling distribution, not through the rubric text itself.

The objective directly optimizes for the property we care about: producing criteria that discriminate. A rubric that causes the judge to score all responses similarly receives zero reward. A rubric that enables discrimination receives reward 1. The judge is frozen throughout training, providing a fixed evaluator against which rubric quality is measured.

2.3 Joint Training

Both the rubric generator and policy model are trained using GRPO (Shao et al., 2024), a variant of policy gradient that computes advantages from group-relative rewards. The policy's GRPO loss is:

$$\mathcal{L}_{\text{policy}}(\theta) = -\mathbb{E}_{q,a\sim\pi_\theta}\left[A(q,a)\log\pi_\theta(a|q)\right] \tag{5}$$

where advantages $A(q, a)$ are derived from judge scores $J(q, \rho_\phi(q), a)$ normalized within each group of 8 responses to the same question. This group normalization removes the need for a learned value function and ensures that only relative quality within a prompt group matters.

Training alternates between two phases: update π_θ for K steps with ρ_ϕ fixed, then update ρ_ϕ for K steps using preference pairs from π_θ. The frequency K controls coupling strength (ablated in Section 3.4).

Why freeze the judge? The judge J is frozen throughout training. This is motivated by the need for a stable reward signal. If the judge were co-trained, apparent improvements in rubric discriminability could reflect the judge learning to agree with the rubric generator rather than the rubrics becoming genuinely more discriminative. The frozen judge provides an invariant measure of rubric quality. The limitation is that the rubric generator can only learn to produce criteria that the frozen judge can interpret and apply; rubrics requiring judgment capabilities beyond the frozen judge's capacity will not be rewarded.

2.4 Constructing Preference Pairs

The rubric generator requires preference pairs, but obtaining them from human annotation would reintroduce the external supervision dependency we aim to eliminate. We instead construct preference pairs from the policy's own outputs.

Replay buffer. We store rollouts with training timestamps and pair current responses with earlier responses to the same question, using the implicit preference that later responses are better. This assumption is the method's central inductive bias. It is not verified with training curves in the present work, and alternative explanations for the replay buffer's effectiveness, such as increased preference pair diversity, cannot be ruled out. The step gap between paired responses controls signal quality.

Inferred question. Given response a^+ to question q, we infer what question it appears to answer ($\hat{q}$) and generate $a^- \sim \pi_\theta(\cdot|\hat{q})$. This provides signal about whether responses address the intended question.

Rubric-conditioned. We generate a^+ conditioned on question and rubric, a^- on question alone. This provides signal about rubric utility: the generator learns to produce criteria that improve responses when provided to the policy.

3 Experiments

Experiments are structured around three questions: Does training rubric generators improve over prompting, and does joint training improve over sequential training? Do learned rubrics generalize across models and judges? What design choices affect rubric and downstream quality?

3.1 Setup

Data. Prompts are drawn from the Tulu 3 preference mixture (Lambert et al., 2024), comprising approximately 271K prompts across general-purpose chat (UltraFeedback, WildChat), instruction following with IFEval-style constraints, mathematical reasoning, coding, scientific literature understanding, and persona-driven synthetic instructions. We filter duplicate prompts from the dataset. Responses are generated on-policy during training as described in Section 2.

Models. Qwen3-8B (Yang et al., 2025) is used as both policy and rubric generator in a parameter-sharing configuration, where a single model handles both roles via different prompts. This configuration minimizes memory requirements; a two-model configuration is ablated in Section 3.4. Qwen3-1.7B serves as the default judge. A key claim is that learned rubrics enable small local judges to provide effective reward signal; using a 1.7B judge tests this directly.

Training. GRPO is used with learning rate 10^{-6}, KL coefficient 0.001, and 8 response samples per question. The alternating frequency $K = 50$ controls how many steps each component trains before switching; this is ablated in Section 3.4. The replay buffer step gap $[20, 100]$ controls the temporal distance between paired responses; this is also ablated in Section 3.4. Other implementation details are in Appendix A.

Baselines. Two prompted baselines isolate the effect of training. *GPT-4.1 prompted*: rubrics generated by prompting GPT-4.1, scored by the same Qwen3-1.7B judge as our method. *Qwen3-8B prompted*: the same architecture as our rubric generator, generating rubrics via prompting without training.

Evaluation. We evaluate along two axes. First, *downstream policy quality*: policies trained with learned rubrics are evaluated on twelve benchmarks spanning mathematical reasoning (GSM8K, MATH), code generation (HumanEval+, MBPP+), general reasoning (BBH, GPQA, ZebraLogic, AGI-Eval), knowledge (MMLU, PopQA), instruction following (IFEval), and open-ended generation (AlpacaEval v3). Second, *rubric quality*: we measure the discriminative accuracy of learned rubrics using RewardBench 2 (Malik et al., 2025) and JudgeBench, which test whether rubric-based evaluation correctly ranks preferred over dispreferred responses across diverse domains.

3.2 Main Comparisons

Prompted baselines. The rubric generator receives no training signal. GPT-4.1 prompted represents a strong upper bound for prompting; Qwen3-8B prompted matches our architecture without training.

Sequential training. The rubric generator is trained first with the policy frozen, learning from preference pairs constructed via inferred question and rubric-conditioned methods (replay buffer requires a changing policy). After rubric training converges, the policy is trained with the rubric generator frozen. This isolates the value of rubric training from the value of joint adaptation.

Co-evolving training. The rubric generator and policy are trained jointly with alternating updates. This enables co-adaptation: as the policy improves, it generates more varied responses that provide richer preference signal; as the rubric generator improves, it provides sharper reward signal. All three preference signal methods including replay buffer can be used.

Table 1: Main comparison of training approaches on downstream policy quality (OLMo3-Adapt benchmarks). HE+ = HumanEval+. All scores are percentages. Rubric discriminative accuracy on held-out benchmarks is reported in Appendix B.

Method	GSM8K	MATH	HE+	MBPP+	BBH	MMLU	IFEval	PopQA	GPQA	Zebra	AGI-E	AlpacaE	Avg
GPT-4.1 (prompted)	95.7	94.2	58.7	65.1	71.7	84.8	34.4	30.7	54.9	81.9	82.9	45.5	66.7
Qwen3-8B (prompted)	95.5	94.6	73.7	58.4	66.8	83.4	35.3	**31.7**	**56.9**	**84.3**	83.9	45.9	67.5
Sequential (IQ)	95.1	**94.7**	65.1	69.0	**74.3**	86.0	30.7	31.3	56.0	83.2	83.2	**52.1**	68.4
Sequential (Rubric)	**96.1**	94.2	70.0	**69.2**	70.1	85.5	32.7	30.8	**56.9**	82.9	83.8	52.0	68.7
Co-evolving (ours)	95.4	94.5	**86.3**	68.5	68.8	**89.6**	**38.1**	30.5	53.8	82.7	**85.5**	40.2	**69.5**

Co-evolving training achieves the highest downstream average (69.5), outperforming both prompted baselines (66.7 for GPT-4.1, 67.5 for Qwen3-8B) and sequential training (68.4 for IQ, 68.7 for Rubric). The gain is concentrated on code generation (HE+ 86.3 vs. 58.7–73.7 for baselines) and general knowledge (MMLU 89.6 vs. 83.4–86.0). Sequential training improves over prompting on some benchmarks (BBH, AlpacaEval) but not others, suggesting that rubric training alone provides partial benefit. The full co-evolving setting, which enables the replay buffer preference signal, yields the strongest overall result. We also measure rubric quality on held-out benchmarks (Appendix B), but these metrics do not predict downstream policy quality. This follows from the training objective: rubrics are optimized for discriminative utility on the policy's own output distribution, not for generalization to arbitrary held-out pairs, consistent with broader findings that held-out reward model quality does not reliably predict downstream RL outcomes (Ivison et al., 2024; Lambert et al., 2024).

3.3 Generalization

A rubric generator has practical value only if it generalizes beyond the training distribution. If learned rubrics work only for the co-trained policy, the generator must be retrained for every new model. If learned rubrics work only with the training judge, the modularity claim fails. We test generalization across policy models and across judges.

3.3.1 Cross-Model Transfer

The rubric generator is trained jointly with a Qwen3-8B policy. We test whether the learned rubric transfers to training a different policy model by using the rubric generator (frozen from the main experiment) to train a Qwen3-14B policy from scratch. If the learned rubric provides effective reward signal for a model not seen during rubric training, it has learned general evaluation criteria rather than Qwen3-8B-specific ones.

The Qwen3-14B policy trained with the learned rubric achieves 75.8 avg across downstream benchmarks, outperforming the Qwen3-8B policy trained with the same rubric (69.5 avg). Gains appear broadly: GSM8K (96.4 vs. 95.4), MATH (95.5 vs. 94.5), HE+ (89.6 vs. 86.3), GPQA (58.0 vs. 53.8), and ZebraLogic (87.5 vs. 82.7). This comparison is confounded by base model capability: Qwen3-14B is a stronger model that may benefit more from any reward signal. Isolating the rubric's contribution would require a Qwen3-14B baseline with prompted rubrics, which we do not run due to compute constraints. The result is nonetheless consistent with the rubric providing general rather than model-specific reward signal.

Table 2: Cross-model transfer. The rubric generator (frozen from the main Qwen3-8B experiment) is used to train a Qwen3-14B policy. Downstream quality measures whether the rubric provides effective reward signal for a model not seen during rubric training. All scores are percentages.

	Downstream Policy Quality (OLMo3-Adapt)												
Configuration	GSM8K	MATH	HE+	MBPP+	BBH	MMLU	IFEval	PopQA	GPQA	Zebra	AGI-E	AlpacaE	Avg
Qwen3-8B + learned rubric (main)	95.4	94.5	86.3	68.5	68.8	89.6	38.1	30.5	53.8	82.7	85.5	40.2	69.5
Qwen3-14B + learned rubric	96.4	95.5	89.6	69.0	69.0	87.7	32.0	–	58.0	87.5	84.3	65.1	75.8

3.3.2 Cross-Judge Evaluation

The rubric generator and judge are designed to be modular: the generator produces criteria, the judge applies them. This modularity is valuable only if rubrics transfer across judges. We test this by training the rubric generator with Qwen3-1.7B and evaluating RewardBench 2 accuracy with judges from different model families at inference time.

Table 3: Cross-judge evaluation. RewardBench 2 (RB2) and JudgeBench (JB) accuracy with different inference-time judges. The rubric generator was trained with Qwen3-1.7B.

Inference Judge	Rubric	RewardBench2							JudgeBench					
		Factuality	Precise IF	Math	Safety	Focus	Ties	Avg	Knowledge	Reasoning	Math	Coding	Overall	Avg
Qwen3-1.7B	Prompted	27.8	26.5	38.1	25.0	30.2	9.7	26.2	27.3	22.4	16.1	11.9	22.3	19.4
(training judge)	Trained	29.3	30.2	37.0	39.1	31.9	33.9	33.6	38.3	20.4	42.9	50.0	35.4	37.9
Qwen3-8B	Prompted	37.4	33.6	**51.5**	**49.9**	45.5	20.3	39.7	48.1	32.7	48.2	**59.5**	45.1	47.1
	Trained	**38.8**	**34.1**	48.9	42.7	**57.1**	44.7	**44.4**	**54.5**	**63.3**	**69.6**	52.4	**59.1**	**60.0**
Llama-3.1-8B	Prompted	30.3	29.4	36.1	31.2	39.1	15.3	30.2	47.4	36.7	55.4	45.2	45.4	46.2
	Trained	28.7	30.4	26.2	40.4	43.3	**52.6**	36.9	52.0	34.7	50.0	38.1	45.1	43.7
Mistral-7B	Prompted	27.8	22.9	28.6	38.5	28.5	34.2	30.1	31.2	17.4	28.6	14.3	24.9	22.8
	Trained	24.6	28.8	30.0	36.0	25.0	33.0	29.6	36.4	19.4	33.9	26.2	30.0	29.0

Trained rubrics transfer well to the same model family: Qwen3-8B with trained rubrics (44.4 RB2 avg, 60.0 JB avg) substantially outperforms prompted rubrics (39.7, 47.1) when the inference judge is Qwen3-8B. Transfer to Llama-3.1-8B is mixed: trained rubrics improve RB2 average (36.9 vs. 30.2) but slightly reduce JB average (43.7 vs. 46.2). For Mistral-7B, trained rubrics provide modest JB gains (29.0 vs. 22.8) but slightly lower RB2 (29.6 vs. 30.1). The strongest gains appear when the inference judge matches or exceeds the training judge's model family, suggesting that learned rubrics partially encode conventions specific to the Qwen model family. Users deploying learned rubrics with out-of-family judges should expect reduced transfer and may benefit from brief fine-tuning of the judge on rubric-formatted inputs.

3.4 Ablations

3.4.1 Alternating Frequency

In co-evolving training, the alternating frequency K controls how many steps each component trains before switching. This choice involves a tradeoff. Frequent switching (K small) tightly couples the models, allowing each to adapt to the other's changes, but may introduce noise if neither converges before switching. Infrequent switching (K large) allows each component to converge but causes distribution shift at transitions: the rubric generator observes responses from a policy that has changed substantially since the last rubric update. We test $K \in \{2, 10, 20, 50, 100\}$ using the co-evolving training setting with replay buffer preference signal. All other hyperparameters remain constant across conditions.

$K = 50$ achieves the best downstream average (69.5), with $K = 10$ second (68.1). Both rubric quality and downstream performance degrade at the extremes: $K = 2$ provides insufficient convergence per phase (68.6 avg), while $K = 100$ allows too much distribution shift between phases (66.7 avg). On rubric quality, $K = 50$ and $K = 100$ achieve similar RB2 averages (33.6 and 33.8), but $K = 50$ leads in JudgeBench (37.9 vs. 35.2), suggesting the intermediate frequency better balances coupling strength with per-phase stability.

Table 4: Effect of alternating frequency K in co-evolving training. All scores are percentages.

	RewardBench2							JudgeBench					
K	Factuality	Precise IF	Math	Safety	Focus	Ties	Avg	Knowledge	Reasoning	Math	Coding	Overall	Avg
2	29.8	30.0	40.6	33.8	31.0	25.3	31.8	19.5	15.3	32.1	33.3	22.0	25.1
10	**30.8**	23.6	**43.5**	36.7	31.0	**34.5**	33.3	25.3	36.7	21.4	26.2	28.0	27.4
20	29.0	25.0	26.5	33.7	**34.9**	13.3	27.1	24.0	21.4	41.1	40.5	28.0	31.8
50 (main)	29.3	30.2	37.0	**39.1**	31.9	33.9	33.6	**38.3**	20.4	42.9	**50.0**	**35.4**	**37.9**
100	29.5	**32.7**	39.5	37.8	31.3	32.2	**33.8**	27.3	**36.7**	26.8	**50.0**	32.6	35.2

	Downstream Policy Quality (OLMo3-Adapt)												
K	GSM8K	MATH	HE+	MBPP+	BBH	MMLU	IFEval	PopQA	GPQA	Zebra	AGI-E	AlpacaE	Avg
2	95.8	94.5	82.4	**68.5**	66.3	85.3	38.6	**33.6**	52.0	81.2	83.1	42.3	68.6
10	**96.4**	94.9	78.0	67.2	62.0	85.1	38.6	31.7	52.5	83.4	83.2	43.9	68.1
20	95.7	**95.0**	62.0	66.3	59.4	84.3	38.6	31.2	52.9	82.1	81.8	41.7	65.9
50 (main)	95.4	94.5	**86.3**	**68.5**	**68.8**	**89.6**	38.1	30.5	**53.8**	82.7	**85.5**	40.2	**69.5**
100	95.5	94.6	68.5	63.0	59.2	83.5	**38.8**	30.5	53.3	**85.3**	80.9	**47.2**	66.7

3.4.2 Model Configuration

Single-model configuration shares parameters between policy and rubric generator, reducing memory but potentially introducing self-evaluation bias: the model may favor criteria that its own responses naturally satisfy. Two-model configuration uses separate parameters, eliminating this bias at the cost of doubled memory. We compare these configurations in the co-evolving setting with replay buffer. The single-model configuration uses one Qwen3-8B model for both roles; the two-model configuration uses separate Qwen3-8B instances. The two-model configuration is evaluated at step 950 (the closest policy checkpoint available) rather than step 1000 due to the different training schedule under parameter separation.

Table 5: Single-model versus two-model configuration. All scores are percentages.

	RewardBench2							JudgeBench					
Configuration	Factuality	Precise IF	Math	Safety	Focus	Ties	Avg	Knowledge	Reasoning	Math	Coding	Overall	Avg
Single-model	29.3	**30.2**	37.0	39.1	**31.9**	33.9	33.6	38.3	20.4	42.9	50.0	35.4	37.9
Two-model	**29.4**	29.9	**37.7**	**40.3**	**31.9**	**34.2**	**33.9**	26.6	**35.7**	33.9	38.1	31.7	33.6

	Downstream Policy Quality (OLMo3-Adapt)												
Configuration	GSM8K	MATH	HE+	MBPP+	BBH	MMLU	IFEval	PopQA	GPQA	Zebra	AGI-E	AlpacaE	Avg
Single-model	95.4	**94.5**	**86.3**	**68.5**	**68.8**	**89.6**	38.1	30.5	**53.8**	82.7	**85.5**	40.2	**69.5**
Two-model	**95.5**	94.4	71.7	66.8	65.3	84.0	**39.0**	**31.5**	**53.8**	**84.5**	82.6	**47.7**	68.1

Single-model achieves a higher downstream average (69.5 vs. 68.1), with the gap driven by code generation (HE+ 86.3 vs. 71.7) and MMLU (89.6 vs. 84.0). Rubric quality metrics are comparable (33.6 vs. 33.9 RB2 avg). The single-model advantage may reflect implicit regularization from shared parameters rather than self-evaluation bias, since the two-model configuration was evaluated at step 950 due to its different training schedule.

3.4.3 Preference Signal Source

The three methods for constructing preference pairs capture different aspects of quality: replay buffer captures general improvement over training, inferred question captures question-addressing, and rubric-conditioned captures rubric utility. These signals could be redundant, with one method dominating, or complementary, with each contributing unique information. We test each signal alone and in combination using the co-evolving training setting. Replay buffer requires a changing policy and is only applicable in co-evolving; inferred question and rubric-conditioned methods can be used in both sequential and co-evolving settings. The combined configurations mix signals equally during training.

Replay buffer dominates on downstream policy quality (69.5 avg), outperforming inferred question (67.7) and rubric-conditioned (67.0) by clear margins. The advantage is largest on code generation (HE+ 86.3 vs. 74.9–75.4) and reasoning (BBH 68.8 vs. 56.4–65.0). Combining signals does not help: Combined (IQ+RC) averages 67.3 and Combined (all 3) averages 68.2, both below replay buffer alone. On rubric quality, replay buffer also leads in JudgeBench (37.9 avg vs. 26.8–30.2), though IQ+RC achieves the highest JB average (41.2) among all configurations. This suggests the

Table 6: Comparison of preference signal sources. All scores are percentages.

	RewardBench2							JudgeBench					
Method	Factuality	Precise IF	Math	Safety	Focus	Ties	Avg	Knowledge	Reasoning	Math	Coding	Overall	Avg
Inferred question	**33.8**	26.6	31.3	36.8	30.5	21.9	30.1	27.9	28.6	26.8	23.8	27.4	26.8
Rubric-conditioned	28.3	24.7	**43.7**	37.2	30.2	24.7	31.5	17.5	37.8	32.1	33.3	27.4	30.2
Replay buffer (main)	29.3	**30.2**	37.0	**39.1**	31.9	**33.9**	**33.6**	**38.3**	20.4	**42.9**	**50.0**	**35.4**	**37.9**
Combined (IQ + RC)	28.1	25.2	35.3	37.9	**32.4**	18.8	29.6	33.8	36.7	39.3	54.8	38.0	41.2
Combined (all 3)	31.6	29.1	34.2	36.2	28.0	24.8	30.6	22.1	21.4	25.0	19.0	22.0	21.9

	Downstream Policy Quality (OLMo3-Adapt)												
Method	GSM8K	MATH	HE+	MBPP+	BBH	MMLU	IFEval	PopQA	GPQA	Zebra	AGI-E	AlpacaE	Avg
Inferred question	**95.8**	94.4	74.9	62.5	65.0	84.1	37.9	31.1	55.4	82.5	82.6	46.6	67.7
Rubric-conditioned	95.7	93.8	75.4	60.8	56.4	84.3	**40.7**	**31.9**	55.4	81.9	82.5	45.4	67.0
Replay buffer (main)	95.4	**94.5**	**86.3**	**68.5**	**68.8**	**89.6**	38.1	30.5	53.8	**82.7**	**85.5**	40.2	**69.5**
Combined (IQ + RC)	95.5	94.4	72.0	63.9	59.3	83.4	38.6	31.3	55.4	82.5	82.9	**48.1**	67.3
Combined (all 3)	95.3	94.3	79.1	64.5	66.4	84.5	37.9	31.2	**56.5**	**82.7**	82.8	43.5	68.2

temporal improvement signal captured by replay buffer provides the strongest rubric training signal, and mixing it with weaker signals dilutes rather than complements.

3.4.4 Judge Size

Larger judges provide more accurate evaluation but increase compute cost. We ablate judge size from 0.6B to 32B parameters, testing whether the rubric generator can learn effectively with judges of varying capacity. All experiments use Qwen3 models of the indicated size as the frozen judge, with Qwen3-8B as the policy and rubric generator in single-model configuration. The co-evolving training setting with replay buffer preference signal is used.

Table 7: Effect of judge model size. All judges use Qwen3 models of the indicated size. The 1.7B judge is the main configuration. All scores are percentages.

	RewardBench2							JudgeBench					
Judge	Factuality	Precise IF	Math	Safety	Focus	Ties	Avg	Knowledge	Reasoning	Math	Coding	Overall	Avg
0.6B	27.3	21.2	29.2	18.4	31.1	24.8	25.3	33.1	29.6	32.1	40.5	32.9	33.8
1.7B (main)	29.3	30.2	37.0	39.1	31.9	**33.9**	33.6	38.3	20.4	42.9	50.0	35.4	37.9
4B	34.9	**35.2**	41.0	40.0	38.2	25.1	35.7	46.1	46.9	48.2	**57.1**	48.0	49.6
8B	36.3	29.8	**50.3**	44.5	39.8	28.7	38.2	44.2	40.8	50.0	54.8	45.4	47.5
14B	**37.1**	32.7	42.8	**45.8**	**42.5**	32.4	**38.9**	43.5	51.0	**62.5**	**57.1**	**50.3**	**53.5**
32B	31.1	24.1	32.9	33.8	33.9	4.3	26.7	**47.4**	**54.1**	55.4	40.5	49.7	49.4

	Downstream Policy Quality (OLMo3-Adapt)												
Judge	GSM8K	MATH	HE+	MBPP+	BBH	MMLU	IFEval	PopQA	GPQA	Zebra	AGI-E	AlpacaE	Avg
0.6B	95.6	94.3	**88.8**	**69.0**	**73.8**	84.9	**39.2**	29.4	54.2	83.6	82.1	**45.2**	**70.0**
1.7B (main)	95.4	**94.5**	86.3	68.5	68.8	**89.6**	38.1	30.5	53.8	82.7	**85.5**	40.2	69.5
4B	**95.8**	94.1	77.4	57.6	63.5	82.1	36.8	33.4	57.4	84.3	84.9	42.7	67.5
8B	95.5	93.3	74.5	56.3	69.2	78.2	37.2	33.5	53.6	**84.8**	83.6	41.9	66.8
14B	95.2	94.4	71.8	58.1	70.3	73.9	39.0	**35.5**	**58.7**	**84.8**	83.6	34.9	66.7
32B	95.6	94.2	79.6	56.2	70.7	78.3	39.0	34.7	**58.7**	84.6	84.2	32.2	67.3

Rubric quality and downstream policy quality exhibit opposite trends with judge size. Rubric quality increases with judge size up to 14B (25.3 $\rightarrow$ 38.9 RB2 avg), as larger judges more reliably apply rubric criteria. However, downstream policy quality *decreases* with judge size: the 0.6B judge produces the best policy (70.0 avg) and the 14B judge the worst (66.7 avg). The gap is especially large on HumanEval+ (88.8 for 0.6B vs. 71.8 for 14B) and BBH (73.8 vs. 70.3).

The 32B judge is anomalous: its rubric quality drops sharply below the 14B judge (26.7 vs. 38.9 RB2 avg), driven by a near-complete collapse on the Ties subcategory (4.3 vs. 32.4 for 14B). This suggests a qualitative failure mode rather than a continuation of the monotonic trend. One likely explanation is that Qwen3-32B defaults to extended thinking tokens that interact poorly with the structured scoring format, collapsing score diversity. Despite this rubric quality collapse, the 32B judge's downstream policy quality (67.3) is comparable to the 4–14B range (66.7–67.5), consistent with the pattern that large judges bypass rubric content and evaluate using internal capabilities.

The primary candidate explanation is rubric dependency: small judges lack the capability to evaluate open-ended responses independently and must rely on rubric criteria, creating a tight feedback loop where better rubrics directly improve the reward signal. Large judges can evaluate using internal knowledge, reducing dependence on rubric content and weakening this pathway. The trained rubrics in Appendix C illustrate the mechanism: the generator learns to pre-solve problems and embed solutions as verifiable criteria, a pattern we call *solution-embedded rubrics*. This reduces evaluation to verification, a task within reach of even a 0.6B judge. A plausible confound is that the effect operates partly through score statistics: small judges may produce higher-variance scores that create more diverse rewards within each GRPO group, improving exploration, while large judges compress the reward distribution. Disentangling these mechanisms requires controlled experiments that vary rubric content while holding score statistics constant, which we leave to future work.

3.4.5 Replay Buffer Age Gap

In co-evolving training with replay buffer, preference pairs are constructed by pairing current responses with earlier responses to the same question. The age gap $[g_{\min}, g_{\max}]$ specifies that rejected responses are sampled uniformly from policy checkpoints $g_{\min}$ to $g_{\max}$ steps in the past. This controls the temporal distance between compared responses.

Small gaps pair similar responses from nearby training steps, which may provide noisy signal because quality differences are minimal. Large gaps pair responses from substantially different policy versions, providing clearer quality differences but risking distribution shift: if the policy has changed dramatically, the earlier responses may not represent realistic negative examples for the current policy. We test three gap ranges: $[5, 20]$, $[20, 100]$ (main), and $[100, 300]$.

Table 8: Effect of replay buffer age gap. The $[20, 100]$ gap is the main configuration. All scores are percentages.

	RewardBench2							JudgeBench					
Age Gap	Factuality	Precise IF	Math	Safety	Focus	Ties	Avg	Knowledge	Reasoning	Math	Coding	Overall	Avg
$[5, 20]$	**30.1**	24.5	**45.4**	34.9	**36.9**	**34.2**	34.3	20.8	25.5	25.0	21.4	22.9	23.2
$[20, 100]$ (main)	29.3	**30.2**	37.0	39.1	31.9	33.9	33.6	**38.3**	20.4	**42.9**	**50.0**	**35.4**	**37.9**
$[100, 300]$	28.6	30.1	44.6	**41.0**	34.5	33.2	**35.3**	17.5	19.4	23.2	40.5	21.7	25.2

	Downstream Policy Quality (OLMo3-Adapt)												
Age Gap	GSM8K	MATH	HE+	MBPP+	BBH	MMLU	IFEval	PopQA	GPQA	Zebra	AGI-E	AlpacaE	Avg
$[5, 20]$	95.1	**94.7**	82.8	**68.7**	59.7	84.4	**39.7**	30.0	51.8	83.1	81.4	**46.0**	68.1
$[20, 100]$ (main)	95.4	94.5	**86.3**	68.5	**68.8**	**89.6**	38.1	30.5	**53.8**	82.7	**85.5**	40.2	**69.5**
$[100, 300]$	**96.1**	94.4	76.6	65.2	57.7	85.3	39.2	**32.2**	50.9	**84.3**	83.7	43.0	67.4

The $[20, 100]$ gap achieves the best downstream average (69.5), outperforming both shorter ($[5, 20]$: 68.1) and longer ($[100, 300]$: 67.4) gaps. Shorter gaps yield noisy preferences that provide weaker rubric training signal (JB avg 23.2 vs. 37.9). Longer gaps provide clearer quality contrasts (slightly higher RB2 avg at 35.3) but the distribution shift between old and current responses reduces downstream effectiveness. The moderate $[20, 100]$ gap balances signal clarity with distributional relevance.

4 Related Work

Reward Models. RLHF trains reward models on human preferences to produce scalar scores (Christiano et al., 2017; Ouyang et al., 2022). Direct preference optimization removes explicit reward modeling (Rafailov et al., 2023). Iterative methods update preferences as policies improve (Xu et al., 2024). These approaches encode evaluation criteria implicitly in model weights. Our rubric generator produces explicit natural language criteria.

LLM-as-a-Judge. Language models evaluate responses when prompted (Zheng et al., 2023; Dubois et al., 2023). Constitutional AI uses self-critique (Bai et al., 2022). RLAIF substitutes AI-generated preferences (Lee et al., 2024). These methods use fixed evaluation prompts. Our rubric generator learns from preference signal which criteria best discriminate response quality.

Rubric-Based Evaluation. Prometheus trains evaluators on rubric-annotated data but obtains rubrics through prompting (Kim et al., 2023). JudgeLM fine-tunes judges with predefined rubrics (Zhu et al., 2023). These treat rubrics as given input. We train rubric generators with a discriminative objective.

Concurrent Rubric-Based RL. Concurrent work applies rubrics to RL training. RaR uses prompted rubrics as fixed structured rewards beyond verifiable domains (Gunjal et al., 2025). RRD recursively refines prompted rubrics with whitened-uniform weighting (Shen et al., 2026). RLCER trains a single model as both policy and rubric generator end-to-end but requires verifiable correctness labels (Sheng et al., 2026). Rubric-ARM alternates training a rubric generator and pairwise judge on offline preference pairs (Xu et al., 2026). Across these concurrent efforts, the recurring tension is between rubric quality and training stability. We resolve it differently: a frozen judge provides stable reward signal while a replay buffer supplies on-policy preference pairs, enabling co-evolution without external labels or verifiers.

Related Approaches. RLAC trains a critic to identify failure points for external verification Wu et al. (2025), reducing verification cost for tasks with automated validators. Our method generates complete evaluation criteria for any task, including those without ground-truth verifiers. WIMHF extracts interpretable features from preference datasets using sparse autoencoders Movva et al. (2025), revealing dataset-level patterns post-hoc. Our rubric generator produces instance-specific criteria at inference time. Self-play (Silver et al., 2017) and self-instruct (Wang et al., 2023) improve policies or data while keeping evaluation fixed. We jointly train the rubric generator with the policy.

5 Conclusion

The core challenge in self-evolving language model training is circular: improving a model requires evaluating it, but reliable evaluation requires either external supervision or a capable judge. We resolve this circularity through factored evaluation: separating rubric generation from judging redirects evaluation quality from judge capability to rubric content, and training the rubric generator with a binary discriminative reward ensures that content is optimized for discrimination rather than surface plausibility. The result is that an 8B model can improve itself using only its own outputs and a 0.6B frozen judge.

The mechanism is solution-embedded rubrics: the generator learns to pre-solve problems and encode solutions as verifiable criteria, reducing evaluation to verification. This explains two findings simultaneously: why small judges produce stronger downstream policies than large ones (verification is within reach of a 0.6B model; open-ended evaluation is not), and why co-evolving training with a replay buffer outperforms sequential training (the policy and rubric generator provide each other with progressively harder targets, sharpening both).

These results reframe reward signal quality in RL for language models. Quality has typically been treated as a property of the judge; the factored architecture shows it is at least partly a property of what the judge is asked to evaluate. A weak judge given precise criteria can outperform a strong judge given vague ones. Whether this principle extends beyond tasks with deterministic answers — to open-ended generation, where rubrics must specify qualitative properties rather than encode solutions — is the boundary condition that the solution-embedded rubric framing makes precise enough to investigate.

REFERENCES

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

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

Yann Dubois, Chen Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy S Liang, and Tatsunori B Hashimoto. Alpacafarm: A simulation framework for methods that learn from human feedback. *Advances in Neural Information Processing Systems*, 36:30039–30069, 2023.

Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Yunzhong He, Bing Liu, and Sean Hendryx. Rubrics as rewards: Reinforcement learning beyond verifiable domains, 2025. URL `https://arxiv.org/abs/2507.17746`.

Shengyi Huang, Jiayi Weng, Rujikorn Charakorn, Min Lin, Zhongwen Xu, and Santiago Ontañón. Cleanba: A reproducible and efficient distributed reinforcement learning platform. *arXiv preprint arXiv:2310.00036*, 2023.

Hamish Ivison, Yizhong Wang, Jiacheng Liu, Zeqiu Wu, Valentina Pyatkin, Nathan Lambert, Noah A. Smith, Yejin Choi, and Hannaneh Hajishirzi. Unpacking dpo and ppo: Disentangling best practices for learning from preference feedback, 2024.

Seungone Kim, Jamin Shin, Yejin Cho, Joel Jang, Shayne Longpre, Hwaran Lee, Sangdoo Yun, Seongjin Shin, Sungdong Kim, James Thorne, et al. Prometheus: Inducing fine-grained evaluation capability in language models. In *The Twelfth International Conference on Learning Representations*, 2023.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. Tülu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

Harrison Lee, Samrat Phatale, Hassan Mansoor, Kellie Ren Lu, Thomas Mesnard, Johan Ferret, Colton Bishop, Ethan Hall, Victor Carbune, and Abhinav Rastogi. RLAIF: Scaling reinforcement learning from human feedback with AI feedback. 2024. URL `https://arxiv.org/abs/2309.00267`.

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

Rajiv Movva, Smitha Milli, Sewon Min, and Emma Pierson. What's in my human feedback? learning interpretable descriptions of preference data. *arXiv preprint arXiv:2510.26202*, 2025.

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

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.

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

William F. Shen, Xinchi Qiu, Chenxi Whitehouse, Lisa Alazraki, Shashwat Goel, Francesco Barbieri, Timon Willi, Akhil Mathur, and Ilias Leontiadis. Rethinking rubric generation for improving llm judge and reward modeling for open-ended tasks, 2026. URL `https://arxiv.org/abs/2602.05125`.

Leheng Sheng, Wenchang Ma, Ruixin Hong, Xiang Wang, An Zhang, and Tat-Seng Chua. Reinforcing chain-of-thought reasoning with self-evolving rubrics, 2026. URL `https://arxiv.org/abs/2602.10885`.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, 2017.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 13484–13508, 2023.

Mian Wu, Gavin Zhang, Sewon Min, Sergey Levine, and Aviral Kumar. RLAC: Reinforcement learning with adversarial critic for free-form generation tasks. *arXiv preprint arXiv:2511.01758*, 2025.

Ran Xu, Tianci Liu, Zihan Dong, Tony Yu, Ilgee Hong, Carl Yang, Linjun Zhang, Tao Zhao, and Haoyu Wang. Alternating reinforcement learning for rubric-based reward modeling in non-verifiable llm post-training, 2026. URL `https://arxiv.org/abs/2602.01511`.

Shusheng Xu, Wei Fu, Jiaxuan Gao, Wenjie Ye, Weilin Liu, Zhiyu Mei, Guangju Wang, Chao Yu, and Yi Wu. Is DPO superior to PPO for LLM alignment? a comprehensive study. *arXiv preprint arXiv:2404.10719*, 2024.

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

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging LLM-as-a-judge with MT-Bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36, 2023.

Lianghui Zhu, Xinggang Wang, and Xinlong Wang. JudgeLM: Fine-tuned large language models are scalable judges. *arXiv preprint arXiv:2310.17631*, 2023.

A Implementation Details

A.1 Model Configuration

- **Policy/Rubric Generator**: Qwen3-8B (8.2B parameters)
- **Rubric Judge**: Qwen3-1.7B (default; ablated from 0.6B to 32B)
- **Parameter Sharing**: Single-model mode (policy and rubric generator share weights)
- **Alternating Frequency**: $K = 50$ steps per training phase

A.2 Training Hyperparameters

Parameter	**Value**
Unique prompts per step	64
Samples per prompt	8
Effective batch size	**512 responses/step**
Per-device batch size	1
Gradient accumulation steps	1

Table 9: Batch configuration. With 64 training GPUs, each GPU processes 8 packed sequences per step.

Batch Configuration.

Parameter	**Value**
Learning rate	1×10^{-6}
LR scheduler	Constant
Warmup steps	0
Optimizer	AdamW (fused)
KL coefficient (β)	0.001
PPO clip range (ϵ)	0.2
KL estimator	KL3

Table 10: Optimization hyperparameters.

Optimization.

Parameter	**Value**
Response length (max)	16,384 tokens
Prompt length (max)	2,048 tokens
Pack length	18,500 tokens
Temperature (policy)	1.0
Temperature (question inference)	0.3
Temperature (rubric judge)	0.0 (greedy)

Table 11: Generation parameters. Long response length supports chain-of-thought reasoning.

Generation.

A.3 Async Training and Active Sampling

Asynchronous Pipeline. We use `async_steps`$= 4$, meaning 4 batches (256 prompts total) are in-flight at any time. The trainer processes batch t while the generator produces batches $t + 1$ through $t + 4$. This overlaps generation and training to maximize GPU utilization. The policy used for generation can be up to 4 steps behind the current training policy, following the Cleanba paradigm (Huang et al., 2023).

Active Sampling. GRPO requires reward variance within each prompt group to compute meaningful advantages. When all 8 samples for a prompt receive identical scores (zero variance), that batch provides no learning signal. With `active_sampling` enabled, such zero-variance batches are filtered and the generator continues sampling until obtaining 64 prompts with non-zero reward variance. This ensures every training step uses informative data.

A.4 Hardware Configuration

- **Training**: 8 nodes $\times$ 8 H100 GPUs = 64 GPUs
- **DeepSpeed**: ZeRO Stage 3 with gradient checkpointing
- **vLLM Engines**: 56 total (40 policy + 16 judge, shared in single-model mode)
- **Tensor Parallelism**: 1 (single GPU per engine)

A.5 Replay Buffer Configuration

- **Buffer size**: 2,048 experiences
- **Age gap**: $[20, 100]$ policy steps (default)
- **Sampling**: Uniform random within valid age range

The age gap ensures rejected samples come from a sufficiently different policy version (at least 20 steps old) while avoiding excessively stale samples (at most 100 steps old). Each experience stores (step, question, answer), and step gap is logged for analysis.

A.6 Checkpointing

- **Save frequency**: Every 25 steps (2 per alternating phase)
- **Retention**: All checkpoints kept (no deletion)
- **Format**: HuggingFace-compatible model weights

B Rubric Quality on Held-Out Benchmarks

The training objective optimizes rubrics for discriminative accuracy on the policy's own output distribution. To test whether this transfers, we evaluate rubric quality on two held-out benchmarks: RewardBench 2 (Malik et al., 2025), which tests discriminative accuracy across diverse domains, and JudgeBench, which tests judge accuracy on expert-annotated examples.

Training consistently improves rubric quality over prompting: co-evolving rubrics (33.6 RB2, 37.9 JB) substantially outperform Qwen3-8B prompted (26.2 RB2, 19.4 JB) and approach GPT-4.1 prompted (36.6 RB2, 41.8 JB). The gap is especially large on JudgeBench (37.9 vs. 19.4 for Qwen3-8B), suggesting that training particularly improves performance on harder, expert-annotated examples.

These held-out metrics do not predict downstream policy quality: GPT-4.1 prompted rubrics achieve the highest discriminative accuracy but produce the weakest policy (Table 1). This is consistent with the broader observation that reward model quality on held-out benchmarks correlates poorly with downstream RL outcomes (Lambert et al., 2024; Malik et al., 2025).

Table 12: Rubric discriminative accuracy on held-out benchmarks. RB2 = RewardBench 2. JB = JudgeBench. All rubric generators use Qwen3-1.7B as the judge. All scores are percentages.

	RewardBench2							JudgeBench					
Method	Factuality	Precise IF	Math	Safety	Focus	Ties	Avg	Knowledge	Reasoning	Math	Coding	Overall	Avg
GPT-4.1 (prompted)	**32.1**	**32.6**	34.4	**39.1**	**35.0**	**46.3**	**36.6**	**41.6**	**50.0**	30.4	45.2	**42.6**	**41.8**
Qwen3-8B (prompted)	27.8	26.5	38.1	25.0	30.2	9.7	26.2	27.3	22.4	16.1	11.9	22.3	19.4
Sequential (IQ)	27.3	30.0	32.1	23.7	28.6	26.0	28.0	24.7	25.5	23.2	33.3	25.7	26.7
Sequential (Rubric)	30.8	31.0	**48.0**	35.5	31.1	38.2	35.8	26.6	23.5	28.6	28.6	26.3	26.8
Co-evolving (ours)	29.3	30.2	37.0	**39.1**	31.9	33.9	33.6	38.3	20.4	**42.9**	**50.0**	35.4	37.9

C Rubric Examples

Representative rubrics from trained (0.6B judge, step 1000) and prompted (Qwen3-8B) generators illustrate qualitative differences.

Example 1: Math Problem

Question: Janet's ducks lay 16 eggs per day. She eats three for breakfast every morning and bakes muffins for her friends every day with four. She sells the remainder at the farmers' market daily for $2 per fresh duck egg. How much does she make every day at the farmers' market?

Trained rubric: (1) Correct calculation: 16 - 3 - 4 = 9 eggs remaining (0.4). (2) Correct multiplication: 9 eggs $\times$ $2 = $18 (0.4). (3) Shows work with intermediate steps (0.2).

Prompted rubric: (1) The response should correctly identify the number of eggs remaining after consumption and baking (0.5). (2) The answer should provide the correct monetary value earned from selling the eggs (0.5).

The trained rubric specifies the exact calculation steps and allocates weight to showing work, while the prompted rubric uses generic phrasing about "correctly identify" and "provide the correct value."

Example 2: Code Generation

Question: Write a Python function to check if a number is prime.

Trained rubric: (1) Handles edge cases: 0, 1, 2 (0.25). (2) Implements correct primality test: checks divisors up to sqrt(n) (0.5). (3) Correct return values: True for prime, False otherwise (0.15). (4) No syntax errors, runs without modification (0.1).

Prompted rubric: (1) The function should correctly implement a prime number check (0.7). (2) The code should handle edge cases appropriately (0.2). (3) The solution should be efficient and well-structured (0.1).

The trained rubric specifies which edge cases and what algorithm, while the prompted rubric uses vague terms like "correctly implement" and "appropriately."