EvoRS: On-Policy Self-Evolution of Reward Systems for Open-Ended Reinforcement Learning

Anonymous authors
Paper under double-blind review

Abstract

Open-ended reinforcement learning often relies on rubric-based rewards for tasks without directly verifiable answers. Yet the policy and reward system form a dynamic feedback loop: as the policy optimizes the current reward, an initially useful reward system may become unreliable due to reward hacking or reduced response discriminability. The reward system should therefore evolve rather than remain fixed during training. Existing dynamic-rubric methods adapt evaluation criteria, but reward failures can also arise from scoring mechanisms or signal composition. We introduce EvoRS, a self-evolving RL framework that evolves the reward system from on-policy experience, representing it as an executable *Reward-DAG*. Specifically, an agentic designer updates this system from on-policy rollouts and reward traces to maintain train-time reliability. Across writing and roleplay, EvoRS achieves the best quality under all three judges, outperforming the policy by 2.107 and 4.767 points, respectively, while reducing reward hacking and coverage failures and preserving reward informativeness. Ablations confirm that a comprehensive fixed reward system cannot remain reliable in open-ended tasks and must evolve throughout training.

1 Introduction

Reinforcement learning (RL) is increasingly used to train LLMs for open-ended tasks, from writing and roleplay to multi-step office workflows handled by work agents (Jia et al., 2025; Tseng et al., 2024). Unlike mathematical reasoning or code generation, for which rewards can often be derived from verifiable answers, these tasks require judgments over multiple interacting dimensions that are difficult to specify. Existing methods therefore rely on rubric-based reward systems, using LLM judges or Generative Reward Models (GRMs) to turn such judgments into scalable training signals (Zheng et al., 2023; Gu et al., 2024; Mahan et al., 2024; Zhang et al., 2025; Gunjal et al., 2025). However, rubric-based reward systems depend on how they are designed and may become less effective during training. This makes reward reliability a central concern in open-ended tasks.

During RL, the policy continually adapts to the current reward system. As the policy changes, a reward system that is reliable for the initial policy may become unreliable later in training. We analyze such failures along three dimensions. **Validity** fails when reward hacking lets the policy obtain high reward without producing the intended behavior (Li et al., 2025; Zhao et al., 2025). **Coverage** becomes insufficient when emerging on-policy responses are not represented by the current reward system (Lang et al., 2024). **Informativeness** diminishes when the reward signal becomes less useful for optimization, for example as rewards saturate or become less separable within same-query groups (Kim et al., 2025b). Together, these failures show that reward reliability cannot be treated as fixed throughout training. The reward system should therefore evolve alongside the policy.

To address these training-time reliability failures, recent dynamic-rubric methods generate rubrics dynamically to cover response patterns missed by fixed rubrics (Yu et al., 2025; Shao et al., 2025). These methods mainly change the rubric criteria, while the scoring mechanisms that turn evaluations into rewards and the composition of multiple signals often remain fixed. Figure 1 illustrates the gap: adding a grounding criterion for a response containing unsupported statistics can still leave its score high under continuous scoring, whereas a penalty mechanism can more selectively penalize such

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

hacking patterns. Reward reliability therefore depends on the complete reward system, including its criteria, scoring mechanisms, and signal composition; these system-level choices should evolve during training rather than remain fixed.

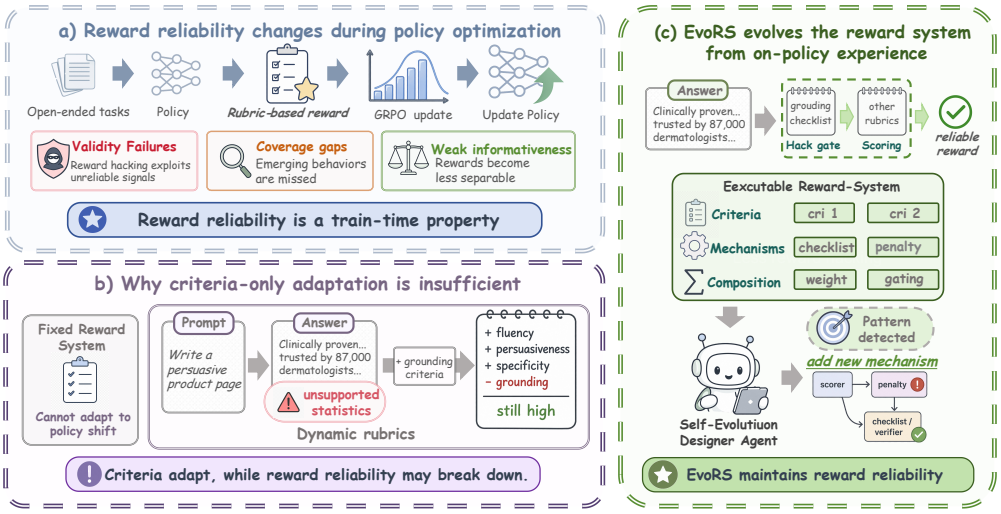


Figure 1: **Reward-system evolution in open-ended RL.** (a) Policy optimization changes reward reliability. (b) Adding criteria alone may leave reward failures unresolved. (c) EvoRS uses on-policy experience to evolve criteria, scoring mechanisms, and composition together.

We therefore introduce EvoRS, a self-evolving reward-system framework for RL on open-ended tasks. EvoRS treats reward design as part of the learning process rather than a one-time specification: the reward system evolves alongside the policy as training exposes new failures. To support this evolution, EvoRS represents the complete reward system as an executable *Reward-DAG*, allowing its criteria, scoring mechanisms, and signal composition to be revised within a unified structure.

We instantiate EvoRS on writing and roleplay, where it achieves the best final benchmark quality under all three judges. While prior methods exhibit increased reward hacking and uneven gains across roleplay dimensions, EvoRS reduces reward-hacking and coverage-failure rates, preserves reward informativeness, and improves all four roleplay dimensions. Component-control ablations further confirm that these gains depend on evolving the reward system alongside the policy throughout training.

Our key contributions are as follows:

- We frame reward reliability in open-ended RL as a training-time problem: as the policy adapts to its reward, the reward system may suffer reward hacking, insufficient on-policy reward coverage, and diminishing reward informativeness.
- We introduce EvoRS, a self-evolving reward-system framework that represents the complete reward system as an executable *Reward-DAG* and searches for bounded candidate states from on-policy evidence. These candidates may revise criteria, scoring mechanisms, or signal composition, while comparative feedback determines the next reward-system state.
- We validate EvoRS on writing and roleplay, showing that it improves final policy quality and reward reliability; controlled ablations further confirm the importance of evolving the reward system alongside the policy.

2 Related Work

2.1 Rubric-Based Rewards and Dynamic Rubrics

Rubric-based rewards use LLM judges or GRMs to provide scalable supervision for open-ended tasks, where quality is multi-dimensional and not directly verifiable. Prior work has instantiated this

supervision through continuous rubric scores and principle-based judgments (Arora et al., 2025; Kim et al., 2025a; Wu et al., 2025; Kim et al., 2024), checklist-based verification (Wei et al., 2025; Lee et al., 2024; Viswanathan et al., 2025), penalty or constraint signals and hybrid designs (Wang et al., 2025; Hong et al., 2026; Liu et al., 2025a), and pairwise preference comparisons (Xu et al., 2025; Jia et al., 2025; Liu et al., 2025b; Du et al., 2026). In RL, rubric-based methods have progressed from broad principles (Bai et al., 2022; Sun et al., 2024; Yu et al., 2025) to task- or query-specific rubrics and anchors (Gunjal et al., 2025; Huang et al., 2025). Recent methods generate or update rubrics dynamically using task information, response comparisons, or online policy outputs (Rezaei et al., 2025; Lv et al., 2026; Xu et al., 2026a; Jia et al., 2026; Shao et al., 2025; Shen et al., 2026; Ding et al., 2026). These methods make the rubric criteria more adaptive, while the scoring and composition procedures that convert them into training rewards generally remain unchanged. This line of work therefore leaves the broader construction of the reward signal outside its scope.

2.2 Reward Reliability under Policy Optimization

Reward reliability becomes more challenging when an evaluator is used as an RL objective. LLM judges and reward models exhibit systematic biases even under static evaluation (Zheng et al., 2023; Ye et al., 2025; Shi et al., 2025; Wang et al., 2026); under optimization, policies can exploit biases, loopholes, or underspecified objectives, leading to reward overoptimization or reward hacking (Skalse et al., 2022; Lambert & Calandra, 2023; Gao et al., 2023; Moskovitz et al., 2024; Hou et al., 2026; Mei et al., 2026). This risk is particularly relevant to rubric-based RL, where explicit criteria can expose shortcut patterns and fixed criteria can miss emerging on-policy responses (Mahmoud et al., 2026; Zhang et al., 2026b; Li et al., 2025; Zhao et al., 2025; Singhal et al., 2024; Lang et al., 2024). Moreover, reward-model accuracy on a fixed distribution does not ensure that the reward remains informative as the policy changes; useful RL signals require meaningful distinctions among on-policy responses, especially within the same query (LeVine et al., 2023; Chen et al., 2024; Razin et al., 2025). Together, these findings show that reward reliability must be evaluated on the responses generated during policy optimization, not only on fixed evaluation outputs.

2.3 Self-Evolving Training and Agent Systems

A broader line of self-evolving training work uses experience to adapt different parts of the learning loop. CoEvolve evolves training data, ECHO evolves a natural-language critic alongside the policy, and ReSkill evolves a skill bank within agentic RL (Yang et al., 2026; Li et al., 2026; He et al., 2026). Other work studies self-evolving harnesses that revise the runtime scaffolding or training environment around an agent; EvoTrainer extends this direction to training-side diagnostics and interventions across policy versions (Chen et al., 2026b; Zhang et al., 2026a; Huang et al., 2026a; Chen et al., 2026a). Across these settings, experience is used to identify what the current training setup lacks and guide targeted, validated changes. Our work applies this principle to the reward system itself: the complete system that supplies the policy's training signal during RL.

3 Motivating Analysis

The central challenge is to keep the reward system reliable as the policy adapts to it. This requires understanding both why a fixed reward system becomes insufficient during training and how on-policy experience can guide its evolution.

The need for reward-system evolution. During RL, the current reward system shapes each policy update, while the updated policy changes the response distribution on which that reward system operates. Figure 2 illustrates this change: a reward system that provides useful discriminability earlier in training may later fail to maintain reward variance and sample-level separability, making it difficult to provide an effective reward signal. Thus, the reward system is not a fixed component that can be determined once before training; it must continue to evolve with the policy.

In rubric-based reward systems for open-ended tasks, a rubric describes what a good response should satisfy, but it does not by itself determine the reward that the policy receives. The same evaluation content can be turned into a reward in different ways: a judge can assign continuous scores, check requirements one by one or deduct points for specific flaws. Table 1 summarizes these

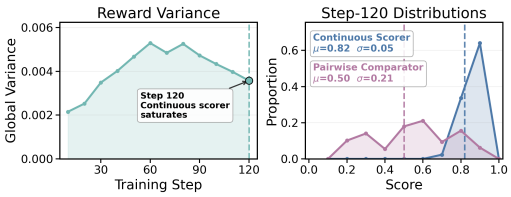


Figure 2: **Fixed scorers can saturate.** Reward variance declines after its mid-training peak, and step-120 continuous scores cluster near the upper end.

Mechanism	Signal	Typical Risk
Continuous	Dense score	Saturation
Checklist	Pass rate	Sparsity
Flaw-Penalty	Penalty	Over-penalty
Pairwise	Preference	Pair dependence

Table 1: **Reward mechanisms expose different signals.** Typical risks include saturation, sparsity, and over-penalization.

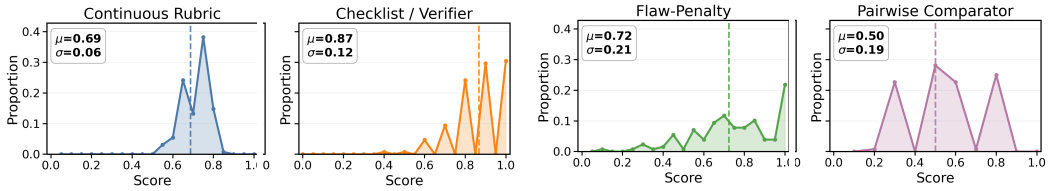


Figure 3: **Mechanisms are not interchangeable.** Different mechanisms induce different score distributions on the same on-policy responses, affecting density, same-query discriminability, and failure sensitivity.

reward mechanisms, and Figure 3 shows that they produce different reward distributions on the same responses. In practice, the mechanisms can also complement one another. For example, a checklist can ensure that necessary requirements are met, while continuous scoring distinguishes quality among responses that pass those checks. Reward reliability therefore depends on the whole path from what is evaluated, to how it is scored, to how multiple signals are combined. What should evolve is not only the rubric or a single scorer, but the complete reward system. Appendix B.2 provides additional analyses of mechanism-specific separability and zero-distinction patterns.

From failure evidence to a targeted repair. Observing a reward failure does not by itself reveal how the reward system should change. For example, low within-query variance may arise because the criteria are too generic to express quality differences among responses, or because scorer saturation compresses different responses into similar scores. The former calls for refining the criteria, whereas the latter calls for changing the scoring mechanism. Reward-system evolution must therefore be based on diagnosis of on-policy rollouts and their reward traces, rather than mapping a single failure indicator directly to an edit. When the evidence does not support a clear diagnosis, the current reward system remains unchanged.

Diagnosis can identify a possible direction for changing the reward system, but it does not establish that the change will improve the reward. Reward-system evolution must also learn from the observed effects of candidate updates. The current and candidate reward systems are evaluated on the same rollout cases to determine whether the candidate repairs the targeted failure while preserving reward behavior that remains useful. This feedback determines which reward-system state is carried into subsequent training and provides experience for later updates. By repeatedly proposing, evaluating, and selecting candidate updates, the reward system can continue to evolve with the policy. Together, diagnosis and comparative validation turn on-policy experience into iterative reward-system evolution. We instantiate this process in EvoRS, described next.

4 Method

We introduce EvoRS, a framework that couples policy learning with reward-system evolution. During RL, the current policy π_t generates groups of responses, which the current reward-system state D_t evaluates to produce both training rewards and node-level reward traces. These responses and traces form the on-policy experience $\mathcal{O}_t$, revealing how the reward system behaves on the policy distribution it currently shapes.

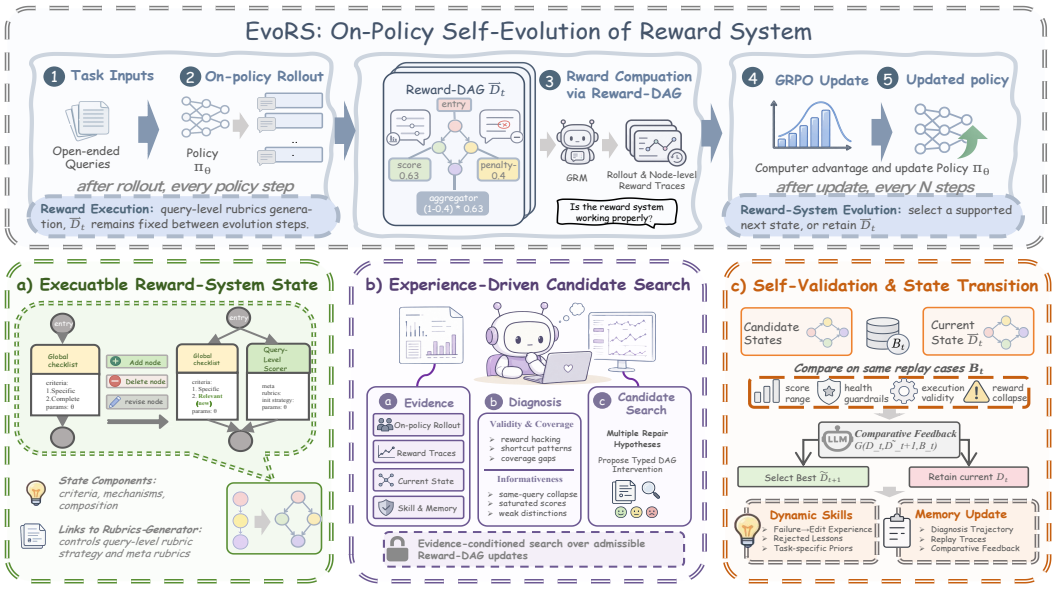


Figure 4: **EvoRS overview.** The top panel couples policy learning with reward-system evolution. The bottom panels show how EvoRS represents the current reward-system state as an executable *Reward-DAG*, generates alternative candidate states from on-policy experience, and uses comparative feedback to select the next state while accumulating memory and dynamic skills for future search.

As training proceeds, EvoRS revisits the reward system every N policy updates and uses this experience to determine whether it should evolve. When an update is needed, it diagnoses the observed evidence, proposes alternative candidate states, and compares their effects through matched replay to select the next reward-system state D_{t+1}. Otherwise, the current state D_t continues to serve the policy. Figure 4 illustrates this closed loop.

4.1 The Reward System as an Evolving State

To make the complete reward system evolvable with the policy, we abstract its current state as an executable *Reward-DAG*:

$$D_t = (\mathcal{V}_t, \mathcal{E}_t, \Theta_t), \tag{1}$$

where $\mathcal{V}_t$ contains executable reward nodes, $\mathcal{E}_t$ specifies their dependencies and execution order, and Θ_t defines the criteria and scoring mechanism used by each node, together with the rules for combining node outputs. Thus, D_t represents in one state what is evaluated, how each reward signal is produced, and how these signals form the final reward. By representing the complete reward system as a *Reward-DAG*, EvoRS turns the reward system itself into an evolvable object.

A Reward-DAG contains Rubric Nodes that produce reward signals and Composition Operators that combine them. Each Rubric Node $v \in \mathcal{V}_t$ instantiates criteria ρ_v and applies a scoring mechanism m_v to produce a node-level signal:

$$h_v = M_{m_v}(x, y; \rho_v, \theta_v, J_\phi). \tag{2}$$

Here, ρ_v may be fixed at the task level or instantiated by $G_\psi(x)$ for the current query. Each node evaluates one response at a time; response groups are used by GRPO and by the diagnostic and comparison procedures described below. The outputs and execution records of all nodes constitute the node-level reward trace introduced above.

A Composition Operator maps upstream node outputs to the scalar reward used by RL:

$$r_D(x, y) = F_{\text{out}}(\{h_v : v \in \text{Pa}(\text{out})\}; \theta_{\text{out}}). \tag{3}$$

It does not evaluate the response again, but determines how existing signals are aggregated, normalized, gated, or routed. Because criteria, scoring mechanisms, and signal combination are represented in the same Reward-DAG, reward-system evolution can modify any of them within a unified representation.

Crucially, Reward-DAG separates reward execution from agentic evolution. The agentic designer uses accumulated on-policy experience only to propose candidate states and does not participate in reward computation itself. Between evolution steps, the active D_t remains fixed. By keeping reward execution independent of the designer, reward computation remains consistent within each training stage and comparable across candidate states. Appendix C.2 provides the execution schema and node-level details; the next section describes how on-policy experience gives rise to candidate reward-system states.

4.2 From On-Policy Experience to Candidate States

To maintain reward-system reliability during training, EvoRS uses an agentic designer to turn accumulated on-policy experience $\mathcal{O}_t$ into a structured diagnosis and a potentially useful revision direction for the current reward-system state D_t. By jointly analyzing sampled responses and their reward traces, the designer relates recurring response patterns to how D_t produces the training signal, then identifies the primary reward failure and its likely source in the Reward-DAG. If the evidence does not support a targeted update, D_t remains unchanged.

Conditioned on the diagnosis, the designer explores alternative repair directions within the admissible update space $\mathcal{A}(D_t)$. It instantiates each direction as a typed update $e_t^{(j)}$ and the corresponding candidate state $\widetilde{D}_{t+1}^{(j)}$:

$$\begin{aligned} \{e_t^{(j)}\}_{j=1}^{M} &= \Pi_{\text{design}}(D_t, \mathcal{O}_t, \mathcal{K}_t), \\ \widetilde{D}_{t+1}^{(j)} &= e_t^{(j)}(D_t), \qquad e_t^{(j)} \in \mathcal{A}(D_t). \end{aligned} \tag{4}$$

Here, $\mathcal{K}_t$ comprises the designer's run-local memory and dynamic skills accumulated from previous evolution steps. Each candidate represents a targeted change to the reward system, but remains a hypothesis until its actual reward behavior is observed. The next section describes how comparative self-validation selects the next reward-system state.

4.3 Self-Validation and Evolutionary State Transition

To turn these hypotheses into an evolution step, EvoRS evaluates alternative reward-system states through the reward behavior they actually produce. It applies the current state D_t and all candidate states $\{\widetilde{D}_{t+1}^{(j)}\}_{j=1}^{M}$ to the same replay cases $\mathcal{B}_t$ drawn from recent on-policy experience. Holding the rollout cases fixed makes their effects directly comparable, revealing whether a candidate addresses the diagnosed failure and how it changes the reward behavior of the current system.

Based on this comparative feedback, EvoRS selects as D_{t+1} the candidate that most consistently corrects the targeted failure cases while preserving rewards on clean cases and the within-query distinctions needed for group-relative policy learning. If no candidate provides a supported improvement, the current state is retained. Each selected, rejected, or no-op trajectory is stored in run-local memory and distilled into dynamic evolution skills, which together form $\mathcal{K}_{t+1}$. By carrying this experience into subsequent diagnoses, the designer progressively adapts how it searches for reward-system improvements. Matched replay therefore provides feedback not only for selecting D_{t+1}, but also for improving the process that generates future candidate states. Appendix C.2 specifies the replay construction and comparison criteria; Appendix C.3 gives the full evolution loop.

5 Experiments and Results

5.1 Experimental Setup

Tasks, data, and evaluation. We evaluate EvoRS on two open-ended domains: writing and roleplay. Writing uses 944 filtered synthetic instructions, while roleplay uses 5,000 queries from MiniMax Role-Play Bench; Appendix D.3 provides the full data construction and distribution. We

Method	Writing				Roleplay (CoSER)							
	Quality	Reliability		Signal	Quality					Reliability		Signal
	WB	HR	CFR	R-Var	Overall	SC	AN	CF	SQ	HR	CFR	R-Var
Base model												
Qwen3-4B base	54.894	7.7	27.8	–	60.909	59.097	64.172	60.487	59.881	6.06	13.0	–
Static rubrics												
RLAIF	56.276	13.8	21.8	0.0010	59.906	59.065	61.889	61.704	56.966	7.43	16.0	0.0029
RaR	56.547	14.7	29.5	0.0014	60.000	60.704	61.752	62.478	55.068	8.52	22.0	0.0061
Dynamic rubrics												
RLER	56.170	11.0	23.9	0.0011	59.621	60.868	59.779	63.364	54.474	9.58	18.0	0.0137
OpenRS	56.427	10.1	31.1	–	58.598	57.456	62.614	59.334	54.986	8.83	15.5	–
Our method												
EvoRS	**57.001**	**6.5**	**19.9**	0.0025	**65.676**	**63.615**	**69.880**	**63.381**	**65.829**	**5.25**	**10.0**	0.0063

Table 2: **Overall policy quality and reward reliability.** We evaluate final policy quality on WritingBench (WB) and CoSER using GPT-5.6-Terra, DeepSeek-V4-Pro, and GLM-5.2 as judges and average their scores; CoSER Overall aggregates its four roleplay dimensions. HR and CFR assess reward validity and coverage, respectively, while R-Var measures within-query reward separation.

evaluate final policy quality using WritingBench (Wu et al., 2025) and CoSER (Wang et al., 2025), with multiple strong judge models on both tasks to reduce dependence on any single evaluator. We report the overall WritingBench score, together with CoSER Overall and its SC, AN, CF, and SQ dimensions. The complete evaluation protocol and judge configurations are provided in Appendix D.6.

Training and reward systems. We compare EvoRS with RLAIF (Bai et al., 2022), RaR (Gunjal et al., 2025), RLER (Shao et al., 2025), and OpenRS (Jia et al., 2026). RLAIF uses a fixed task-level rubric, while RaR uses fixed query-specific rubrics. RLER augments each task rubric with additional criteria elicited from current policy responses, whereas OpenRS constructs pairwise adaptive rubrics from response comparisons. All methods use the same GRPO pipeline, training data, rollout budget, and evaluation protocol. They use the same local Qwen3.5-27B model as the GRM for rubric-based reward computation. The reward system of EvoRS starts from the same single-node continuous reward system as RLAIF and evolves every five RL steps. The appendices provide the full training-data distribution and experimental configurations (Appendices D.3 and C.1).

Metrics and audits. We use WritingBench and CoSER to evaluate final response quality. Following prior analyses and designs of exploit rates and reward variance, we evaluate reward reliability from three dimensions: HR measures the occurrence of predefined reward-hacking patterns (Mahmoud et al., 2026; Thaman, 2026), CFR measures predefined hard failures that the reward may under-cover (Zhou et al., 2023; Jiang et al., 2024), and R-Var measures within-query reward separation (Razin et al., 2025; Xu et al., 2026b). The HR/CFR categories are constructed before method comparison and applied blindly to all outputs. Appendix D.5 provides details.

5.2 Main Results

Table 2 compares different methods in terms of final policy quality and reward reliability. On WritingBench, all RL methods improve over the base policy, while EvoRS produces the largest quality gain of 2.107 points and is the only method that reduces HR below the base level. The contrast is sharper on CoSER. Methods trained with fixed or partially adapted reward systems do not improve overall quality over the base policy and exhibit higher HR. Their improvements are also uneven across dimensions: for example, RaR improves SC and CF but reduces AN and SQ. This pattern is consistent with overoptimization of an increasingly exploitable reward signal—the policy learns the aspects emphasized by the current reward while sacrificing behaviors it fails to capture. In contrast, EvoRS improves CoSER Overall by 4.767 points over the base policy and ranks first across all four dimensions.

These results demonstrate the benefit of **maintaining a reward system that remains effective for the evolving policy rather than continuing to optimize fixed reward criteria**. The next section

examines how reward reliability changes throughout training. Appendix D.6 provides judge-specific results, paired uncertainty estimates, and further analysis of the roleplay outputs. A three-seed WritingBench stability experiment is also reported there.

6 Analysis

6.1 The Need for Full, Training-Time Reward-System Evolution

Reward reliability requires evolving the complete reward system. We restrict EvoRS to rubric-criterion updates, yielding a criteria-only evolution variant that leaves scoring mechanisms and signal composition fixed. Its CoSER score drops from 65.676 to 61.155 (Table 3). Adapting reward criteria alone therefore cannot maintain reliable reward signals as policy behavior changes in open-ended RL.

Reward composition alone cannot replace co-evolution. We further take the final *Reward-DAG* obtained at the end of a full EvoRS run and use it unchanged from the start of a separate training run (Fixed final *Reward-DAG*). Its CoSER score falls to 60.573 (Table 3). Thus, the gain is not explained solely by discovering a better combination of criteria, scoring mechanisms, and signals; the reward system must continue to evolve with the policy throughout training.

Variant	WB	Δ	CoSER	Δ
Full EvoRS	**57.001**	–	**65.676**	–
Only criteria-level adaptation	56.593	−0.408	61.155	−4.521
Fixed final *Reward-DAG*	56.308	−0.693	60.573	−5.103

Table 3: **Scope and timing of reward-system evolution.**

6.2 Reward-System Evolution under Policy Shift

Figure 5 traces reward validity, coverage, and informativeness as the policy changes, allowing us to examine whether newly exposed reward failures are followed by corresponding changes in the reward-system state.

Validity and coverage. In writing, reward-hacking patterns increase under both static and dynamic-rubric baselines, whereas EvoRS remains lower and declines further after the annotated reward-system transition. In roleplay, EvoRS similarly maintains a lower CFR and shows a further reduction after the step-45 coverage repair. Because the audit categories are hidden from the designer, these trajectories reflect failures discovered from on-policy experience rather than direct optimization of the reported metrics. Appendix E.5 provides detailed examples of these diagnoses and state changes.

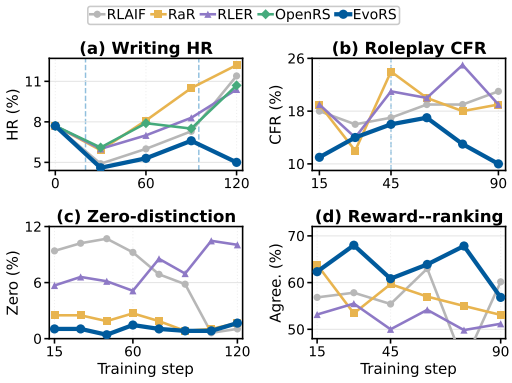


Figure 5: **Training-time reward dynamics.** EvoRS reduces writing HR and roleplay CFR after reward-system updates (a,b), while maintaining same-query distinctions and preference alignment (c,d).

Informativeness. Figure 5(c,d) complements the final R-Var results by examining whether within-query distinctions remain both available and meaningful during training. Across the 480-group audit, EvoRS maintains a low Zero-distinction rate and stronger agreement with blind same-query preference rankings across most checkpoints. The evolved reward therefore does not merely produce more dispersed scores; its distinctions remain aligned with independent response preferences. Appendix E.4 provides the evaluation protocol and reliability checks.

6.3 Generality across Tasks, Policies, and Reward Models

Beyond open-ended language generation, we find that EvoRS remains effective in the broader setting of general agentic RL. We perform multi-turn agentic RL training with Qwen3.5-8B in office-work scenarios and evaluate the resulting agents on OfficeBench. As shown in Table 4, EvoRS achieves 40.67, outperforming both the base policy (36.33) and RaR (39.67). These results support the effectiveness of reward-system evolution in tool-using agentic tasks. Appendix D.3 provides the training-data construction and reward setup.

Method	OfficeBench	Δ
Qwen3.5-8B base	36.33	–
RaR	39.67	+3.34
EvoRS	**40.67**	**+4.34**

Table 4: **Tool-using agentic RL.** Qwen3.5-8B agents are trained on multi-turn office tasks and evaluated on OfficeBench; Δ is relative to the base policy.

We next examine whether the benefit of reward-system evolution depends on the Qwen3-4B policy or the Qwen3.5-27B reward model used in the main experiments. We scale the policy to Qwen3-8B (Yang et al., 2025) and evaluate both Qwen3.5-27B and RewardAnything-8B (Yu et al., 2025) as GRMs. As shown in Table 5, EvoRS outperforms RaR under both reward models and also improves over the Qwen3-8B base policy. The advantage of reward-system evolution therefore persists across policy scales and GRM families.

6.4 Self-Evolution Component Ablations

We now ablate the capabilities that support the search over improved reward-system states.

Table 6 examines how EvoRS searches for its next reward-system state. Candidate selection compares alternative repair directions through their observed reward behavior; update rejection preserves the current state when no candidate provides an improvement; and evolution memory makes previous outcomes available to later search. Removing any of these capabilities reduces final quality, with substantially larger effects on multi-turn roleplay. The dimension-level results in Appendix E.1 further show that some variants retain high SC or CF while sharply reducing AN and SQ, indicating less balanced optimization.

Designer dependence. We also test whether EvoRS requires GPT-5.4 as its designer. In roleplay training, three designers that correctly use the diagnostic and editing tools achieve CoSER scores of 64.34–65.93, whereas a configuration with frequent edit failures scores 61.44 (Appendix Table 11). Thus, EvoRS is not tied to GPT-5.4, but requires reliable tool use.

Method	GRM	CoSER
Qwen3-8B base	–	64.545
RaR	Qwen3.5-27B	62.044
EvoRS	Qwen3.5-27B	**66.370**
RaR	RewardAnything-8B	62.761
EvoRS	RewardAnything-8B	**65.660**

Table 5: **Policy/GRM generality.** CoSER Overall, averaged over three judges.

Variant	WB	Δ	CoSER	Δ
Full EvoRS	**57.001**	–	**65.676**	–
w/o selection	56.959	−0.042	62.494	−3.182
w/o rejection	56.807	−0.194	63.911	−1.765
w/o memory/skills	56.918	−0.083	62.412	−3.264

Table 6: **Self-evolution component ablations.** Three-judge averages; Δ is relative to full EvoRS.

7 Conclusion

We presented EvoRS, a framework for on-policy self-evolution of reward systems in open-ended RL. It represents the active reward system as an executable *Reward-DAG*, diagnoses its failures from on-policy reward traces, and searches for bounded candidate states,closing the loop between policy learning and reward-system evolution.

Our experiments show that dynamic-rubric methods adapt evaluation criteria but leave scoring mechanisms and signal composition fixed, yielding uneven gains. EvoRS instead uses on-policy failures to revise the complete reward system throughout training.

To support reproducibility, we provide anonymized supplementary materials covering the full experimental pipeline; their contents are detailed in Appendix D.1.

8 AI USE STATEMENT

We used generative AI to synthesize training instructions and task- and query-level rubrics, which form part of the training data and reward setup. We also used AI assistants for language polishing and code development. The authors reviewed the generated materials and remain responsible for the research design, experiments, analyses, claims, and final manuscript content.

REFERENCES

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

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Kamile Lukosuite, Liane Lovitt, Michael Sellitto, Nelson Elhage, Nicholas Schiefer, Noemi Mercado, Nova DasSarma, Robert Lasenby, Robin Larson, Sam Ringer, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Tamera Lanham, Timothy Telleen-Lawton, Tom Conerly, Tom Henighan, Tristan Hume, Samuel R. Bowman, Zac Hatfield-Dodds, Ben Mann, Dario Amodei, Nicholas Joseph, Sam McCandlish, Tom Brown, and Jared Kaplan. Constitutional ai: Harmlessness from ai feedback, 2022. URL https://arxiv.org/abs/2212.08073.

Chao Chen, Chengzu Li, Zhiwei Li, Yinhong Liu, and Zhijiang Guo. From trainee to trainer: Llm-designed training environment for rl with multi-agent reasoning, 2026a. URL https://arxiv.org/abs/2606.17682.

Guhong Chen, Yingcheng Shi, Yongbin Li, Binhua Li, Xander Xu, Hu Wei, Shiwen Ni, Min Yang, and Jieping Ye. Evotrainer: Co-evolving llm policies and training harnesses for autonomous agentic reinforcement learning, 2026b. URL https://arxiv.org/abs/2606.03108.

Yanjun Chen, Dawei Zhu, Yirong Sun, Xinghao Chen, Wei Zhang, and Xiaoyu Shen. The accuracy paradox in rlhf: When better reward models don't yield better language models. *arXiv preprint arXiv:2410.06554*, 2024.

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

Chengyu Du, Xintao Wang, Aili Chen, Weiyuan Li, Rui Xu, Junteng Liu, Zishan Huang, Rong Tian, Zijun Sun, Yuhao Li, Liheng Feng, Deming Ding, Pengyu Zhao, and Yanghua Xiao. Her: Human-like reasoning and reinforcement learning for llm role-playing, 2026. URL https://arxiv.org/abs/2601.21459.

Mingxuan Du, Benfeng Xu, Chiwei Zhu, Xiaorui Wang, and Zhendong Mao. Deepresearch bench: A comprehensive benchmark for deep research agents, 2025. URL https://arxiv.org/abs/2506.11763.

Daniel Fein, Sebastian Russo, Violet Xiang, Kabir Jolly, Rafael Rafailov, and Nick Haber. Litbench: A benchmark and dataset for reliable evaluation of creative writing, 2025. URL https://arxiv.org/abs/2507.00769.

Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Yuanzhuo Wang, Wen Gao, Lionel Ni, and Jian Guo. A survey on LLM-as-a-judge, 2024. URL https://arxiv.org/abs/2411.15594.

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.

Zelin He, Haotian Lin, Boran Han, Wei Zhu, Haoyang Fang, Bernie Wang, Xuan Zhu, Runze Li, and Matthew Reimherr. Reskill: Reconciling skill creation with policy optimization in agentic rl, 2026. URL https://arxiv.org/abs/2606.01619.

Yihan Hong, Huaiyuan Yao, Bolin Shen, Wanpeng Xu, Hua Wei, and Yushun Dong. Rulers: Locked rubrics and evidence-anchored scoring for robust llm evaluation. *arXiv preprint arXiv:2601.08654*, 2026. URL https://arxiv.org/abs/2601.08654.

Or Honovich, Leshem Choshen, Roee Aharoni, Ella Neeman, Idan Szpektor, and Omri Abend. q^2: Evaluating factual consistency in knowledge-grounded dialogues via question generation and question answering. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 7856–7870. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.emnlp-main.619. URL https://aclanthology.org/2021.emnlp-main.619/.

Hongru Hou, Tiehua Mei, Denghui Geng, Jinhui Huang, Ao Xu, Hengrui Chen, Jiaqing Liang, and Deqing Yang. ProRL: Effective reinforcement learning for proactive recommendation via rectified policy gradient estimation. *arXiv preprint arXiv:2605.28293*, 2026.

Chengsong Huang, Zifeng Wang, Rujun Han, Jun Yan, Yanfei Chen, Zoey CuiZhu, Ke Jiang, Peng Xia, Han Yu, Yufan Zhuang, Yifei Ming, Jiaqi Pan, Bhavana Dalvi Mishra, Jiaxin Huang, Burak Gokturk, Tomas Pfister, and Chen-Yu Lee. Envharness: Awakening static worlds for agent learning, 2026a. URL https://arxiv.org/abs/2608.19880.

Tianyi Huang, Nathan Huang, Justin Tang, Wenqian Chen, and Elsa Fan. Permutation-consensus listwise judging for robust factuality evaluation, 2026b. URL https://arxiv.org/abs/2603.20562.

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, Jianguo Li, and Junbo Zhao. Reinforcement learning with rubric anchors, 2025. URL https://arxiv.org/abs/2508.12790.

Ruipeng Jia, Yunyi Yang, Yongbo Gai, Kai Luo, Shihao Huang, Jianhe Lin, Xiaoxi Jiang, and Guanjun Jiang. Writing-zero: Bridge the gap between non-verifiable tasks and verifiable rewards, 2025. URL https://arxiv.org/abs/2506.00103.

Ruipeng Jia, Yunyi Yang, Yuxin Wu, Yongbo Gai, Siyuan Tao, Mengyu Zhou, Jianhe Lin, Xiaoxi Jiang, and Guanjun Jiang. Open rubric system: Scaling reinforcement learning with pairwise adaptive rubric, 2026. URL https://arxiv.org/abs/2602.14069.

Yuxin Jiang, Yufei Wang, Xingshan Zeng, Wanjun Zhong, Liangyou Li, Fei Mi, Lifeng Shang, Xin Jiang, Qun Liu, and Wei Wang. Followbench: A multi-level fine-grained constraints following benchmark for large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024. URL https://arxiv.org/abs/2310.20410.

Seungone Kim, Jamin Shin, Yejin Choi, Joel Jang, Shayne Longpre, Hwaran Lee, Sangdoo Yun, Seongjin Shin, Sungdong Kim, James Thorne, and Minjoon Seo. Prometheus: Inducing fine-grained evaluation capability in language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=8euJaTveKw.

Seungone Kim, Juyoung Suk, Ji Yong Cho, Shayne Longpre, Chaeeun Kim, Dongkeun Yoon, Guijin Son, Yejin Cho, Sheikh Shafayat, Jinheon Baek, Sue Hyun Park, Hyeonbin Hwang, Jinkyung Jo, Hyowon Cho, Haebin Shin, Seongyun Lee, Hanseok Oh, Noah Lee, Namgyu Ho, Se June Joo, Miyoung Ko, Yoonjoo Lee, Hyungjoo Chae, Jamin Shin, Joel Jang, Seonghyeon Ye, Bill Yuchen Lin, Sean Welleck, Graham Neubig, Moontae Lee, Kyungjae Lee, and Minjoon Seo. The biggen bench: A principled benchmark for fine-grained evaluation of language models with language models, 2025a. URL https://arxiv.org/abs/2406.05761.

Sunghwan Kim, Dongjin Kang, Taeyoon Kwon, Hyungjoo Chae, Dongha Lee, and Jinyoung Yeo. Rethinking reward model evaluation through the lens of reward overoptimization. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13252–13280. Association for Computational Linguistics, 2025b. URL https://aclanthology.org/2025.acl-long.649.

Philippe Laban, Tobias Schnabel, Paul N. Bennett, and Marti A. Hearst. SummaC: Re-visiting NLI-based models for inconsistency detection in summarization. *Transactions of the Association for Computational Linguistics*, 10:163–177, 2022. doi: 10.1162/tacl_a_00453. URL https://aclanthology.org/2022.tacl-1.10/.

Nathan Lambert and Roberto Calandra. The alignment ceiling: Objective mismatch in reinforcement learning from human feedback. *arXiv preprint arXiv:2311.00168*, 2023.

Hunter Lang, Fei Huang, and Yongbin Li. Fine-tuning language models with reward learning on policy. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2024. URL https://aclanthology.org/2024.naacl-long.75.

Yukyung Lee, Joonghoon Kim, Jaehee Kim, Hyowon Cho, and Pilsung Kang. Checkeval: Robust evaluation framework using large language model via checklist. *arXiv preprint arXiv:2403.18771*, 2024. URL https://arxiv.org/abs/2403.18771.

Will LeVine, Benjamin Pikus, Anthony Chen, and Sean Hendryx. A baseline analysis of reward models' ability to accurately analyze foundation models under distribution shift. *arXiv preprint arXiv:2311.14743*, 2023.

Weiyuan Li, Xintao Wang, Siyu Yuan, Rui Xu, Jiangjie Chen, Qingqing Dong, Yanghua Xiao, and Deqing Yang. Curse of knowledge: Your guidance and provided knowledge are biasing LLM judges in complex evaluation. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 14900–14924, Suzhou, China, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-emnlp.805. URL https://aclanthology.org/2025.findings-emnlp.805/.

Zhicong Li, Lingjie Jiang, Yulan Hu, Xingchen Zeng, Yixia Li, Xiangwen Zhang, Guanhua Chen, Zheng Pan, Xin Li, and Yong Liu. No more stale feedback: Co-evolving critics for open-world agent learning, 2026. URL https://arxiv.org/abs/2601.06794.

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

Zijun Liu, Peiyi Wang, Runxin Xu, Shirong Ma, Chong Ruan, Peng Li, Yang Liu, and Yu Wu. Inference-time scaling for generalist reward modeling. *arXiv preprint arXiv:2504.02495*, 2025b. URL https://arxiv.org/abs/2504.02495.

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

Dakota Mahan, Duy Van Phung, Rafael Rafailov, Chase Blagden, Nathan Lile, Louis Castricato, Jan-Philipp Fränken, Chelsea Finn, and Alon Albalak. Generative reward models, 2024. URL https://arxiv.org/abs/2410.12832.

Anas Mahmoud, MohammadHossein Rezaei, Zihao Wang, Anisha Gunjal, Bing Liu, and Yunzhong He. Reward hacking in rubric-based reinforcement learning. *arXiv preprint arXiv:2605.12474*, 2026.

Tiehua Mei, Minxuan Lv, Leiyu Pan, Zhenpeng Su, Hongru Hou, Hengrui Chen, Ao Xu, and Deqing Yang. Good reasoning makes good demonstrations: Implicit reasoning quality supervision via in-context reinforcement learning. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 30452–30469, 2026.

Sewon Min, Kalpesh Krishna, Xinxi Lyu, Mike Lewis, Wen tau Yih, Pang Wei Koh, Mohit Iyyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. Factscore: Fine-grained atomic evaluation of factual precision in long form text generation, 2023. URL https://arxiv.org/abs/2305.14251.

Ted Moskovitz, Aaditya K. Singh, DJ Strouse, Tuomas Sandholm, Ruslan Salakhutdinov, Anca D. Dragan, and Stephen McAleer. Confronting reward model overoptimization with constrained rlhf. In *International Conference on Learning Representations*, 2024.

Noam Razin, Zixuan Wang, Hubert Strauss, Stanley Wei, Jason Lee, and Sanjeev Arora. What makes a reward model a good teacher? an optimization perspective. In *Advances in Neural Information Processing Systems*, volume 38, pp. 59162–59222. Curran Associates, Inc., 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/554e056fe2b6d9fd27ffcd3367ae1267-Paper-Conference.pdf.

MohammadHossein Rezaei, Robert Vacareanu, Zihao Wang, Clinton Wang, Yunzhong He, and Afra Feyza Akyürek. Online rubrics elicitation from pairwise comparisons. *arXiv preprint arXiv:2510.07284*, 2025.

Rulin Shao, Akari Asai, Shannon Zejiang Shen, Hamish Ivison, Varsha Kishore, Jingming Zhuo, Xinran Zhao, Molly Park, Samuel G. Finlayson, David Sontag, Tyler Murray, Sewon Min, Pradeep Dasigi, Luca Soldaini, Faeze Brahman, Wen-tau Yih, Tongshuang Wu, Luke Zettlemoyer, Yoon Kim, Hannaneh Hajishirzi, and Pang Wei Koh. Dr tulu: Reinforcement learning with evolving rubrics for deep research, 2025. URL https://arxiv.org/abs/2511.19399.

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

Lin Shi, Chiyu Ma, Wenhua Liang, Xingjian Diao, Weicheng Ma, and Soroush Vosoughi. Judging the judges: A systematic study of position bias in llm-as-a-judge. In *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, pp. 292–314, Mumbai, India, 2025. The Asian Federation of Natural Language Processing and The Association for Computational Linguistics. doi: 10.18653/v1/2025.ijcnlp-long.18. URL https://aclanthology.org/2025.ijcnlp-long.18/.

Prasann Singhal, Tanya Goyal, Jiacheng Xu, and Greg Durrett. A long way to go: Investigating length correlations in rlhf. In *Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=G8LaO1P0xv.

Joar Max Viktor Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward gaming. In *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=yb3HOXO3lX2.

Zhiqing Sun, Yikang Shen, Hongxin Zhang, Qinhong Zhou, Zhenfang Chen, David Cox, Yiming Yang, and Chuang Gan. Salmon: Self-alignment with instructable reward models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/dda5cac5272a9bcd4bc73d90bc725ef1-Abstract-Conference.html.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, S. H. Cai, Yuan Cao, Y. Charles, H. S. Che, Cheng Chen, Guanduo Chen, Huarong Chen, Jia Chen, Jiahao Chen, Jianlong Chen, Jun Chen, Kefan Chen, Liang Chen, Ruijue Chen, Xinhao Chen, Yanru Chen, Yanxu Chen, Yicun Chen, Yimin Chen, Yingjiang Chen, Yuankun Chen, Yujie Chen, Yutian Chen, Zhirong Chen, Ziwei Chen, Dazhi Cheng, Minghan Chu, Jialei Cui, Jiaqi Deng, Muxi Diao, Hao Ding, Mengfan Dong, Mengnan Dong, Yuxin Dong, Yuhao Dong, Angang Du, Chenzhuang Du, Dikang Du, Lingxiao Du, Yulun Du, Yu Fan, Shengjun Fang, Qiulin Feng, Yichen Feng, Garimugai Fu, Kelin Fu, Hongcheng Gao, Tong Gao, Yuyao Ge, Shangyi Geng, Chengyang Gong, Xiaochen Gong, Zhuoma Gongque, Qizheng Gu, Xinran Gu, Yicheng Gu, Longyu Guan, Yuanying Guo, Xiaoru Hao, Weiran He, Wenyang He, Yunjia He, Chao Hong, Hao Hu, Jiaxi Hu, Yangyang Hu, Zhenxing Hu, Ke Huang, Ruiyuan Huang, Weixiao Huang, Zhiqi Huang, Tao Jiang, Zhejun Jiang, Xinyi Jin, Yu Jing, Guokun Lai, Aidi Li, C. Li, Cheng Li, Fang Li, Guanghe Li, Guanyu Li, Haitao Li,

Haoyang Li, Jia Li, Jingwei Li, Junxiong Li, Lincan Li, Mo Li, Weihong Li, Wentao Li, Xinhang Li, Xinhao Li, Yang Li, Yanhao Li, Yiwei Li, Yuxiao Li, Zhaowei Li, Zheming Li, Weilong Liao, Jiawei Lin, Xiaohan Lin, Zhishan Lin, Zichao Lin, Cheng Liu, Chenyu Liu, Hongzhang Liu, Liang Liu, Shaowei Liu, Shudong Liu, Shuran Liu, Tianwei Liu, Tianyu Liu, Weizhou Liu, Xiangyan Liu, Yangyang Liu, Yanming Liu, Yibo Liu, Yuanxin Liu, Yue Liu, Zhengying Liu, Zhongnuo Liu, Enzhe Lu, Haoyu Lu, Zhiyuan Lu, Junyu Luo, Tongxu Luo, Yashuo Luo, Long Ma, Yingwei Ma, Shaoguang Mao, Yuan Mei, Xin Men, Fanqing Meng, Zhiyong Meng, Yibo Miao, Minqing Ni, Kun Ouyang, Siyuan Pan, Bo Pang, Yuchao Qian, Ruoyu Qin, Zeyu Qin, Jiezhong Qiu, Bowen Qu, Zeyu Shang, Youbo Shao, Tianxiao Shen, Zhennan Shen, Juanfeng Shi, Lidong Shi, Shengyuan Shi, Feifan Song, Pengwei Song, Tianhui Song, Xiaoxi Song, Hongjin Su, Jianlin Su, Zhaochen Su, Lin Sui, Jinsong Sun, Junyao Sun, Tongyu Sun, Flood Sung, Yunpeng Tai, Chuning Tang, Heyi Tang, Xiaojuan Tang, Zhengyang Tang, Jiawen Tao, Shiyuan Teng, Chaoran Tian, Pengfei Tian, Ao Wang, Bowen Wang, Chensi Wang, Chuang Wang, Congcong Wang, Dingkun Wang, Dinglu Wang, Dongliang Wang, Feng Wang, Hailong Wang, Haiming Wang, Hengzhi Wang, Huaqing Wang, Hui Wang, Jiahao Wang, Jinhong Wang, Jiuzheng Wang, Kaixin Wang, Linian Wang, Qibin Wang, Shengjie Wang, Shuyi Wang, Si Wang, Wei Wang, Xiaochen Wang, Xinyuan Wang, Yao Wang, Yejie Wang, Yipu Wang, Yiqin Wang, Yucheng Wang, Yuzhi Wang, Zhaoji Wang, Zhaowei Wang, Zhengtao Wang, Zhexu Wang, Zihan Wang, Zizhe Wang, Chu Wei, Ming Wei, Chuan Wen, Zichen Wen, Chengjie Wu, Haoning Wu, Junyan Wu, Rucong Wu, Wenhao Wu, Yuefeng Wu, Yuhao Wu, Yuxin Wu, Zijian Wu, Chenjun Xiao, Jin Xie, Xiaotong Xie, Yuchong Xie, Yifei Xin, Bowei Xing, Boyu Xu, Jianfan Xu, Jing Xu, Jinjing Xu, L. H. Xu, Lin Xu, Suting Xu, Weixin Xu, Xinbo Xu, Xinran Xu, Yangchuan Xu, Yichang Xu, Yuemeng Xu, Zelai Xu, Ziyao Xu, Junjie Yan, Yuzi Yan, Guangyao Yang, Hao Yang, Junwei Yang, Kai Yang, Ningyuan Yang, Ruihan Yang, Xiaofei Yang, Xinlong Yang, Ying Yang, Yi Yang, Yi Yang, Zhen Yang, Zhilin Yang, Zonghan Yang, Haotian Yao, Dan Ye, Wenjie Ye, Zhuorui Ye, Bohong Yin, Chengzhen Yu, Longhui Yu, Tao Yu, Tianxiang Yu, Enming Yuan, Mengjie Yuan, Xiaokun Yuan, Yang Yue, Weihao Zeng, Dunyuan Zha, Haobing Zhan, Dehao Zhang, Hao Zhang, Jin Zhang, Puqi Zhang, Qiao Zhang, Rui Zhang, Xiaobin Zhang, Y. Zhang, Yadong Zhang, Yangkun Zhang, Yichi Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yushun Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Chenguang Zhao, Feifan Zhao, Jinxiang Zhao, Shuai Zhao, Xiangyu Zhao, Yikai Zhao, Zijia Zhao, Huabin Zheng, Ruihan Zheng, Shaojie Zheng, Tengyang Zheng, Junfeng Zhong, Longguang Zhong, Weiming Zhong, M. Zhou, Runjie Zhou, Xinyu Zhou, Zaida Zhou, Jinguo Zhu, Liya Zhu, Xinhao Zhu, Yuxuan Zhu, Zhen Zhu, Jingze Zhuang, Weiyu Zhuang, Ying Zou, and Xinxing Zu. Kimi k2.5: Visual agentic intelligence, 2026. URL https://arxiv.org/abs/2602.02276.

Tencent WorkBuddy Bench Team. Tencent workbuddy bench: A multi-domain coding-agent benchmark with contamination-resistant task construction, 2026. URL https://arxiv.org/abs/2607.20911.

Kunvar Thaman. Reward hacking benchmark: Measuring exploits in llm agents with tool use. *arXiv preprint arXiv:2605.02964*, 2026. URL https://arxiv.org/abs/2605.02964.

Yu-Min Tseng, Yu-Chao Huang, Teng-Yun Hsiao, Wei-Lin Chen, Chao-Wei Huang, Yu Meng, and Yun-Nung Chen. Two tales of persona in LLMs: A survey of role-playing and personalization. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 16612–16631, Miami, Florida, USA, 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.findings-emnlp.969/.

Vijay Viswanathan, Yanchao Sun, Xiang Kong, Meng Cao, Graham Neubig, and Sherry Wu. Checklists are better than reward models for aligning language models. In *Advances in Neural Information Processing Systems*, volume 38, pp. 114728–114754. Curran Associates, Inc., 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/a6837c1dd021f76f1b4098e3722052a8-Paper-Conference.pdf.

Alex Wang, Kyunghyun Cho, and Mike Lewis. Asking and answering questions to evaluate the factual consistency of summaries. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 5008–5020. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.acl-main.450. URL https://aclanthology.org/2020.acl-main.450/.

Atticus Wang, Iván Arcuschin, and Arthur Conmy. Automatically finding reward model biases. *arXiv preprint arXiv:2602.15222*, 2026.

Xintao Wang, Heng Wang, Yifei Zhang, Xinfeng Yuan, Rui Xu, Jen-tse Huang, Siyu Yuan, Haoran Guo, Jiangjie Chen, Wei Wang, Yanghua Xiao, and Shuchang Zhou. Coser: Coordinating llm-based persona simulation of established roles. *arXiv preprint arXiv:2502.09082*, 2025. URL https://arxiv.org/abs/2502.09082.

Zilong Wang, Yuedong Cui, Li Zhong, Zimin Zhang, Da Yin, Bill Yuchen Lin, and Jingbo Shang. Officebench: Benchmarking language agents across multiple applications for office automation, 2024. URL https://arxiv.org/abs/2407.19056.

Jerry Wei, Chengrun Yang, Xinying Song, Yifeng Lu, Nathan Hu, Jie Huang, Dustin Tran, Daiyi Peng, Ruibo Liu, Da Huang, Cosmo Du, and Quoc V. Le. Long-form factuality in large language models, 2024. URL https://arxiv.org/abs/2403.18802.

Tianjun Wei, Wei Wen, Ruizhi Qiao, Xing Sun, and Jianghong Ma. Rocketeval: Efficient automated llm evaluation via grading checklist. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=zJjzNj6QUe.

Yuning Wu, Jiahao Mei, Ming Yan, Chenliang Li, Shaopeng Lai, Yuran Ren, Zijia Wang, Ji Zhang, Mengyue Wu, Qin Jin, and Fei Huang. Writingbench: A comprehensive benchmark for generative writing. In *Advances in Neural Information Processing Systems*, volume 38. Curran Associates, Inc., 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/4aedf0cba303537fcb6cf948bb41b2df-Paper-Datasets_and_Benchmarks_Track.pdf.

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

Wenyuan Xu, Xiaochen Zuo, Chao Xin, Yu Yue, Lin Yan, and Yonghui Wu. A unified pairwise framework for rlhf: Bridging generative reward modeling and policy optimization. *arXiv preprint arXiv:2504.04950*, 2025.

Yixuan Even Xu, Yash Savani, Fei Fang, and J. Zico Kolter. Not all rollouts are useful: Down-sampling rollouts in llm reinforcement learning. *Transactions on Machine Learning Research*, 2026b. URL https://arxiv.org/abs/2504.13818.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Shidong Yang, Ziyu Ma, Tongwen Huang, Yiming Hu, Yong Wang, and Xiangxiang Chu. Coevolve: Training llm agents via agent-data mutual evolution, 2026. URL https://arxiv.org/abs/2604.15840.

Sihui Yang, Keping Bi, Wanqing Cui, Jiafeng Guo, and Xueqi Cheng. Linkage: Listwise ranking among varied-quality references for non-factoid qa evaluation via llms, 2024. URL https://arxiv.org/abs/2409.14744.

Jiayi Ye, Yanbo Wang, Yue Huang, Dongping Chen, Qihui Zhang, Nuno Moniz, Tian Gao, Werner Geyer, Chao Huang, Pin-Yu Chen, Nitesh V. Chawla, and Xiangliang Zhang. Justice or prejudice? quantifying biases in llm-as-a-judge. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/fdca08d371e4b6c031397909e20043bd-Abstract-Conference.html.

Zhuohao Yu, Jiali Zeng, Weizheng Gu, Yidong Wang, Jindong Wang, Fandong Meng, Jie Zhou, Yue Zhang, Shikun Zhang, and Wei Ye. Rewardanything: Generalizable principle-following reward models, 2025. URL https://arxiv.org/abs/2506.03637.

Hangfan Zhang, Shao Zhang, Kangcong Li, Chen Zhang, Yang Chen, Yiqun Zhang, Lei Bai, and Shuyue Hu. Self-harness: Harnesses that improve themselves, 2026a. URL https://arxiv.org/abs/2606.09498.

Junkai Zhang, Zihao Wang, Lin Gui, Swarnashree Mysore Sathyendra, Jaehwan Jeong, Victor Veitch, Wei Wang, Yunzhong He, Bing Liu, and Lifeng Jin. Chasing the tail: Effective rubric-based reward modeling for large language model post-training. In *The Fourteenth International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=pBjy4ek2QV.

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/214308a2d5e3f83ef9ad2739e1cbc46d-Abstract-Conference.html.

Yulai Zhao, Haolin Liu, Dian Yu, Sunyuan Kung, Meijia Chen, Haitao Mi, and Dong Yu. One token to fool LLM-as-a-judge, 2025. URL https://arxiv.org/abs/2507.08794.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2306.05685.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023. URL https://arxiv.org/abs/2311.07911.

Appendix topic	What is covered	Purpose	Details
Evaluation Protocols	Multi-judge WritingBench and CoSER protocols, common-success aggregation, judge-specific results, paired uncertainty estimates, roleplay output analysis, and blinded HR/CFR audits.	Documents how benchmark quality, robustness diagnostics, judge variance, and audit blinding are handled.	App. D.5; App. D.6
Training-Time Evidence	Selected and non-selected candidate states, reward-variance dynamics, and blind reward-ranking audits over same-query rollout groups.	Makes reward-system state transitions traceable and separates final-output gains from process-level evolution evidence.	App. E.2; App. E.4
Implementation and Reproducibility	Model roles, training settings, prompts, Reward-DAG schema, typed edit operations, legality checks, candidate-comparison thresholds, runtime cost, and designer tools/skills.	Specifies the reward-system interface and constrained designer workflow used in the experiments.	App. C.1; App. C.2; App. C.4; App. C.5
Baselines and Task Setup	Artifact use, data safety, writing and roleplay training data, initial rubric construction, unified GRPO setup, and reward-system instantiations of static and dynamic-rubric baselines.	Clarifies the shared experimental setting and how baseline reward systems are instantiated within it.	App. D.2; App. D.3; App. D.4
Case Studies	Writing guard-style penalty repair and roleplay coverage repair, including the diagnosed rollout evidence and selected Reward-DAG states.	Illustrates how EvoRS turns on-policy reward failures into targeted candidate states.	App. E.5

Table 7: Roadmap of appendix materials for evaluation protocols, training-time evidence, implementation details, baselines, and case studies.

A Appendix Roadmap and Robustness Checks

Because the evaluation involves open-ended RL, LLM judges, and a self-evolving reward-system loop, we include additional checks for evaluation robustness, auditor blinding, reward-evolution traceability, baseline instantiation, and reproducibility. Table 7 gives a roadmap to these materials. It is intended as a navigation aid rather than as additional experimental evidence.

B Supplementary Motivating Analysis

B.1 Reward Mechanism Taxonomy

This subsection clarifies the reward-mechanism space used in the motivating analysis. We use *reward mechanism* to denote the scoring or judgment procedure that turns criteria and response evidence into an optimization signal. A mechanism specifies the elicited judgment, the granularity of the produced signal, and its normalization into a reward or preference signal. This is distinct from the *criterion source*, which determines what evidence or criteria are evaluated, and from the *composition operator*, which determines how multiple local signals are aggregated, gated, or routed.

This distinction is important for interpreting dynamic-rubric methods. Query-level rubrics, retrieved references, and generated checklists change the criterion source. Weighted sums, product guards, and routing modules change composition. A reward mechanism changes how judgments over criteria are elicited and exposed to the policy. Thus two reward systems can evaluate similar criteria but induce different training signals, for example by using dense scalar scores, pass-rate vectors, penalty scores, pairwise preferences, listwise ranks, or claim-level support vectors.

Table 8 summarizes the mechanisms considered in this work and representative related work. The table is intended as a taxonomy of common scoring protocols rather than a claim that all mechanisms are used as live RL rewards in our experiments.

Continuous / pointwise rubric scorer. The most common rubric-based protocol asks a judge to assign scalar scores to a response, either globally or along multiple rubric dimensions. With criteria

Mechanism	Output signal	Typical granularity	Best suited for	Representative references
Continuous / pointwise rubric scorer	Dense scalar or dimension-wise scalar	Response or rubric-dimension level	Broad open-ended quality, writing quality, general rubric scoring	BiGGen Bench, WritingBench, Prometheus, DeepSeek-GRM (Kim et al., 2025a; Wu et al., 2025; Kim et al., 2024; Liu et al., 2025b)
Checklist / verifier scorer	Pass/fail vector, pass rate, or partial-credit pass rate	Checklist-item or criterion level	Explicit requirements, constraint satisfaction, coverage of necessary elements	RocketEval, CheckEval, RLCF, HealthBench, Rubrics as Rewards (Wei et al., 2025; Lee et al., 2024; Viswanathan et al., 2025; Arora et al., 2025; Gunjal et al., 2025)
Flaw-penalty / deduction scorer	Downward penalty score with violation trace	Flaw, defect, or violation level	Sparse bad-tail failures, roleplay violations, reward-hacking artifacts	CoSER for penalty-based roleplay judging; reward-hacking work as motivation (Wang et al., 2025; Mahmoud et al., 2026; Zhang et al., 2026b)
Pairwise preference comparator	Pairwise winner, preference label, or win-rate reward	Response-pair level	Same-query relative quality and subtle writing or roleplay differences	MT-Bench / Chatbot Arena, pairwise RLHF, Writing-Zero, HER, LitBench (Zheng et al., 2023; Xu et al., 2025; Jia et al., 2025; Du et al., 2026; Fein et al., 2025)
Listwise / ranking scorer	Full ranking, top-k, or rank-normalized score	Response-set level	Ordering a set of candidate responses under the same query	LINKAGE and permutation-consensus listwise judging (Yang et al., 2024; Huang et al., 2026b)
Claim-level factuality / grounding scorer	Claim-support vector, factual precision, unsupported-claim rate, or groundedness score	Atomic-claim, QA-probe, citation, or evidence-unit level	Long-form factuality, summarization consistency, RAG, deep research, citation grounding	FActScore, SAFE/LongFact, QAGS, Q^2, SummaC, DeepResearch Bench (Min et al., 2023; Wei et al., 2024; Wang et al., 2020; Honovich et al., 2021; Laban et al., 2022; Du et al., 2025)

Table 8: Representative reward mechanisms for open-ended and partially grounded generation. The taxonomy is organized by scoring protocol and output signal, not by whether a paper uses the word "rubric" or "reward". Criterion-source adaptation methods are discussed separately from reward mechanisms.

$\{c_k\}_{k=1}^K$, a typical normalized score is

$$h_{\text{cont}}(x, y) = \frac{1}{K} \sum_{k=1}^{K} s_k, \qquad s_k \in [0, 1].$$

This dense signal is easy to scalarize for RL and covers broad quality dimensions, but it can saturate as the policy improves, reducing same-query separability. Fine-grained LLM evaluation benchmarks and reward models commonly instantiate this mechanism through dimension-wise or rubric-conditioned scalar scores (Kim et al., 2025a; Wu et al., 2025; Kim et al., 2024; Liu et al., 2025b). Evidence-anchored robust scoring systems such as RULERS are also related, but they should be viewed as robust pointwise scoring rather than penalty mechanisms (Hong et al., 2026).

Checklist / verifier scorer. Checklist mechanisms decompose the evaluation into explicit requirements and expose item-level satisfaction:

$$h_{\text{check}}(x, y) = \frac{1}{K} \sum_{k=1}^{K} z_k, \qquad z_k \in \{0, 1\},$$

with graded variants allowing $z_k \in \{0, 0.5, 1\}$. This produces an interpretable pass vector or pass rate and is useful when necessary conditions must be checked explicitly. RocketEval and CheckEval study checklist-based LLM evaluation, RLCF uses checklist-style rewards for alignment, and HealthBench uses physician-written criteria that are judged as met or unmet before aggregation (Wei et al., 2025; Lee et al., 2024; Viswanathan et al., 2025; Arora et al., 2025). Rubrics as Rewards provides an RL example of structured rubric rewards beyond verifiable domains (Gunjal et al., 2025).

Flaw-penalty / deduction scorer. Deduction mechanisms start from an implicit or explicit full score and subtract penalties for detected flaws:

$$h_{\text{pen}}(x, y) = \text{clip}\left(1 - \lambda \sum_{j} \alpha_j v_j, 0, 1\right),$$

where v_j denotes violation severity and α_j denotes the corresponding weight. Unlike checklist scoring, the output is a downward correction targeted at sparse failures. CoSER is a direct example in roleplay evaluation: it identifies roleplay mismatch cases such as role-setting, language-style, action, or psychological mismatches and uses penalty-based LLM judging (Wang et al., 2025). Reward-hacking analyses motivate such bad-tail mechanisms, although they are not themselves deduction scorers (Mahmoud et al., 2026; Zhang et al., 2026b).

Pairwise preference comparator. Pairwise mechanisms elicit relative judgments between two responses:

$$p_{ij} = J_\phi(x, y_i, y_j),$$
$$h_{\text{pair}}(x, y_i) = \frac{1}{K - 1} \sum_{j \neq i} \mathbf{1}[p_{ij} = i].$$

The output can be a winner label, a preference probability, or a win-rate reward. This mechanism is often more stable than absolute scoring when candidates differ subtly under the same prompt, but it is sensitive to position and order bias and is more expensive than pointwise scoring. Pairwise judging is used in LLM-as-judge settings such as MT-Bench / Chatbot Arena and in pairwise RLHF or generative reward-modeling formulations (Zheng et al., 2023; Xu et al., 2025). Writing-Zero and HER provide open-ended writing and roleplay examples, while LitBench directly supports creative-writing comparison as a benchmark signal (Jia et al., 2025; Du et al., 2026; Fein et al., 2025).

Listwise / ranking scorer. Listwise mechanisms compare a candidate set at once and return a ranking or top-k selection:

$$\text{rank}_i = J_\phi(x, \{y_1, \ldots, y_K\}),$$
$$h_{\text{rank}}(x, y_i) = 1 - \frac{\text{rank}_i - 1}{K - 1}.$$

This can strengthen same-query discrimination by asking the judge to order all candidates jointly. It is distinct from a best-of-K winner selector, which is a top-1 special case of listwise evaluation. The main risks are context length, set-composition dependence, and permutation sensitivity. LINKAGE provides a direct listwise-ranking formulation for non-factoid QA evaluation, and permutation-consensus listwise judging studies robustness issues for listwise factuality evaluation (Yang et al., 2024; Huang et al., 2026b).

Claim-level factuality / grounding scorer. For grounded open-ended generation, a mechanism can first extract atomic claims, QA probes, or citation-grounded statements and then judge support:

$$\mathcal{C}(y) = \{c_1, \ldots, c_M\}, \qquad z_j = V_\phi(c_j; x, \mathcal{S}_x),$$

$$h_{\text{fact}}(x, y) = \frac{1}{M} \sum_{j=1}^{M} \mathbf{1}[z_j = \text{supported}].$$

Here $\mathcal{S}_x$ denotes source documents, dialogue context, retrieved evidence, or cited sources. This mechanism differs from checklist scoring because the evaluated units are extracted from the response rather than fixed in advance. It is useful for long-form factuality, summarization consistency, RAG-style answers, and deep research reports. FActScore and SAFE/LongFact evaluate long-form factuality through atomic facts and search-augmented verification; QAGS, Q^2, and SummaC use QA- or entailment-style probes for factual consistency; DeepResearch Bench extends this idea to citation accuracy and effective citation counts in deep research agents (Min et al., 2023; Wei et al., 2024; Wang et al., 2020; Honovich et al., 2021; Laban et al., 2022; Du et al., 2025).

Boundary with criterion-source adaptation. Several related methods change what is evaluated rather than how scoring is elicited. Query-level rubric generation, online rubric elicitation from pairwise comparisons, rubric-anchor methods, and dynamic-rubric RL are therefore best viewed as criterion-source or reward-system adaptation methods, not as separate scoring mechanisms by themselves (Liu et al., 2025a; Huang et al., 2025; Xu et al., 2026a; Rezaei et al., 2025; Lv et al., 2026; Shao et al., 2025; Jia et al., 2026). In our framework, such criterion-source changes can coexist with mechanism changes inside the same executable *Reward-DAG*.

B.2 Additional Motivating Results

This subsection provides three supplementary analyses that complement the main-text distribution and training-dynamics figures. All numbers are computed from cached offline mechanism scores; no additional judge calls were made for this summary. The analyses cover: (i) query-level versus task-level criteria under the same scoring mechanism, (ii) mechanism behavior across response buckets, and (iii) mechanism-specific score-distribution shift during training.

Query-level versus task-level criteria. This comparison isolates criterion-source specialization from reward-mechanism changes. Both conditions use the same continuous rubric scoring path over the same 128 writing responses; the only difference is whether the scorer uses task-level global criteria or query-level specialized criteria. Query-level criteria slightly lower the global mean score, from 0.687 to 0.678, but increase score dispersion and within-query separability: global standard deviation rises from 0.061 to 0.074, same-query range rises from 0.064 to 0.093, and same-query standard deviation rises from 0.026 to 0.037. The paired score correlation remains high, with Pearson 0.825 and Spearman 0.823, showing that query-level criteria preserve the broad scoring trend while reshaping local distinctions.

Mechanism behavior by response type. We next bucket the same writing responses into five behavior categories, with 20 examples per bucket: normal high-quality outputs, under-complete or constraint-missing outputs, reward-hacking-like or fabricated-authority outputs, template or scaffold leakage, and subtly better same-query candidates selected by pointwise quality. Figure 7 shows that the mechanisms are complementary rather than interchangeable. Continuous scoring tracks broad quality, but remains relatively high on fabricated-authority cases (0.709). Checklist scores sharply drop on explicit incompleteness or constraint misses (0.717), but remain high on fabricated-authority and scaffold-leakage cases (0.958 and 0.942). Flaw-penalty scoring is most sensitive to bad-tail artifacts, dropping to 0.423 for fabricated-authority cases and 0.513 for scaffold leakage. Pairwise

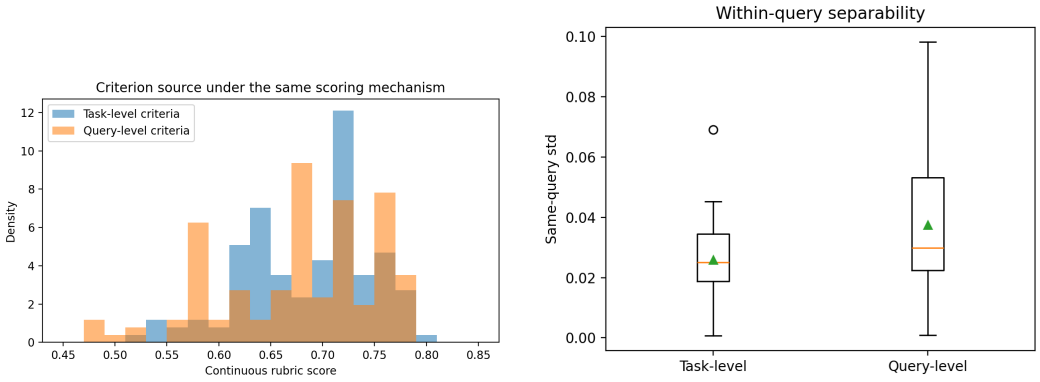


Figure 6: Query-level versus task-level criteria under the same continuous scoring mechanism. Query-level criteria change the reward landscape and improve within-query separability, but this is a criterion-source change rather than a new reward mechanism.

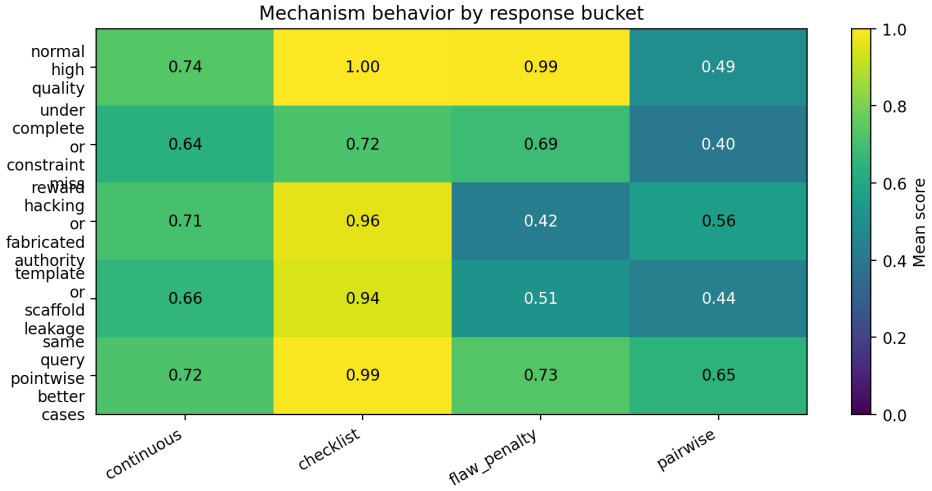


Figure 7: Mechanism behavior by response type. Continuous scoring, checklist verification, flaw-penalty detection, and pairwise comparison expose different failure modes and therefore provide complementary reward signals.

scores are centered near 0.5 over arbitrary buckets, as expected, but rise to 0.646 on independently selected same-query better cases.

Distribution shift during training. Finally, we compare step-0 base-model responses with step-120 trained responses using the same offline mechanism scoring pipeline. Figure 8 shows the clearest shifts for the RaR checkpoint. Continuous rubric scores move upward from mean 0.645 at step 0 to 0.785 at step 120, while checklist scores move from 0.878 to 0.945. The checklist signal also becomes less informative within groups: same-query standard deviation drops from 0.100 to 0.050, and zero-variance group rate rises from 0.188 to 0.531. Continuous scoring retains more local spread under RaR than checklist, with same-query standard deviation increasing from 0.024 to 0.038 and zero-variance group rate decreasing from 0.500 to 0.344. These results illustrate that reward mechanisms can shift or saturate differently as the policy distribution changes, so final reward variance should be separated from base positive-reward informativeness and bad-tail penalty activation.

Category	Setting
Policy initialization	Qwen3-4B.
Training algorithm	GRPO using the same trainer for all reward-system variants.
Reward/scoring model	Local Qwen3.5-27B GRM; scalar rewards normalized to $[0, 1]$.
Designer model	Codex SDK with GPT-5.4, invoked every 5 RL steps for EvoRS.
Benchmark judges	GPT-5.6-Terra, DeepSeek-V4-Pro, and GLM-5.2; scores are equally averaged.
Training length	120 GRPO steps for the main reported runs.
Batching	32 prompts per step, 4 rollouts per prompt.
Sequence lengths	4096-token prompts; 4096-token writing responses and 2048-token role-play responses.
Optimization	Learning rate 1×10^{-6}, PPO mini-batch size 4, micro-batch size 2 per GPU, low-variance KL with coefficient 0.001, entropy coefficient 0.
Checkpointing	Save every 5 steps.
Hardware	8 NVIDIA A800 GPUs for each main training run.
RL training budget	About 24 hours wall-clock on average per main run, i.e., about 192 A800 GPU-hours.

Table 9: Model, compute, and core hyperparameter settings for the main GRPO runs. Proprietary designer and evaluation models have undisclosed parameter counts, so we report their names and roles instead.

C Method Details

C.1 Implementation Details for Reward Evolution

Model, compute, and training settings. Table 9 summarizes the model and training settings used for the main reported GRPO runs. The policy model is Qwen3-4B (Yang et al., 2025), and the local GRM/rubric-updater model is Qwen3.5-27B. GPT-5.4 is used by the reward-system designer. Final benchmark quality is evaluated by GPT-5.6-Terra, DeepSeek-V4-Pro, and GLM-5.2; additional auditor and ranking analyses use the models specified in their corresponding subsections. These proprietary models do not expose public parameter counts, so we report their model names and roles. We use a fixed GRPO recipe across methods rather than method-specific hyperparameter search. All compared reward-system variants share the same policy initialization, training data, rollout budget, batching, optimizer settings, and checkpoint schedule; reward-evolution thresholds and update cadences are fixed before the final comparison.

Runtime cost and agent overhead. The RL training budget above counts the training path. Offline benchmark scoring, HR/CFR audits, blind ranking annotations, and multi-judge benchmark evaluation are not included in the RL training GPU-hour accounting. Online reward scoring during RL is included.

We report writing-side wall-clock cost from the training logs. The per-step time is the logged `perf/time_per_step`, which includes rollout generation, reward scoring, log-probability and reference-policy computation, actor update, checkpointing when triggered, and hook overhead. For EvoRS, this logged step time already includes the designer update when the update hook runs. We therefore also report the training-only step time after subtracting `meta_judge/elapsed_s` on update steps and the average agent time per reward-evolution step.

Method	Parsed steps	Avg. step time	Avg. agent update time	Agent interval
RLAIF	120	511.3s	–	–
RaR	120	663.8s	–	–
RLER	120	718.4s	–	–
OpenRS	120	1037.4s	–	–
EvoRS, training only	120	753.0s	–	–
EvoRS, with designer	120	801.5s	242.8s	5 steps

Table 10: Writing-side runtime cost parsed from training logs. For EvoRS, the designer runs every 5 GRPO steps; its average evolution-step time is 242.8s. The training-only row subtracts designer elapsed time from update steps, while the with-designer row reports the logged end-to-end step time.

GRM scoring and query-level criterion instantiation. All pointwise scalar rewards in the main experiments are computed with a local Qwen3.5-27B GRM. The scorer receives the query, the assistant response, and the active rubric or *Reward-DAG* node specification, and returns item-level scores together with a normalized scalar reward in $[0, 1]$. For continuous rubric nodes, the final node score is the mean or weighted mean of item scores after parsing and clipping. For checklist or verifier nodes, boolean or partial-credit outcomes are normalized into a pass-rate score. For flaw-penalty nodes, detected violations produce a downward correction that is combined with the positive reward through the composition operator. When a query-level rubric node is active, the same local Qwen3.5-27B model instantiates query-aware criteria under the node's fixed generation policy. This is part of reward execution and does not change the active reward-system state.

Continuous GRM scoring prompt. The continuous rubric scorer uses the following runtime prompt template. Long query, response, and rubric fields are abbreviated here, but the scoring instructions and output schema match the implementation.

Continuous GRM Scoring Prompt

```
You are a severe but fair reward model grader.

You will receive a query, an assistant response, and a list of rubric items.
Score each rubric item independently on a 0.0-1.0 scale.

Use degree of satisfaction, not a yes/no interpretation:
- 0.00-0.20: absent, contradicted, or harmful.
- 0.20-0.40: very weak; only superficial or accidental evidence.
- 0.40-0.55: partial but flawed; important requirements are missing.
- 0.55-0.70: adequate but generic, thin, or uneven.
- 0.70-0.84: clearly good; specific evidence supports the criterion.
- 0.84-0.92: very strong; robust, nuanced, and well grounded.
- 0.92-1.00: exceptional; rare, highly effective execution.

Calibration and ceiling rules:
- Do not give high scores for merely avoiding errors.
- Generic or interchangeable responses should usually be capped at 0.65.
- Scores above 0.72 require concrete evidence from the response.
- Scores above 0.84 require strong craft, insight, grounding, or reader effect.
- If the response exposes chain-of-thought or private reasoning, cap the final
  mean score at 0.35.

Strict scoring addendum:
- Start from 0.55 and move up or down based on concrete evidence.
- 0.70 is already a good score; above 0.70 requires specific non-generic merit.
- 0.80+ should be stronger than normal good answers; 0.88+ should be rare.
- If the rationale for an item would be generic, cap that item at 0.68.
- Polished surface without insight, specificity, or task judgment caps at 0.65.
- Separate responses by depth, specificity, judgment, and reader effect.

Query:
{query}

Assistant response:
{response}

Rubric items:
{rubric_items}

Return JSON only:
{
  "item_results": [
    {"score": 0.58, "reason": "brief reason with concrete evidence"}
  ],
```

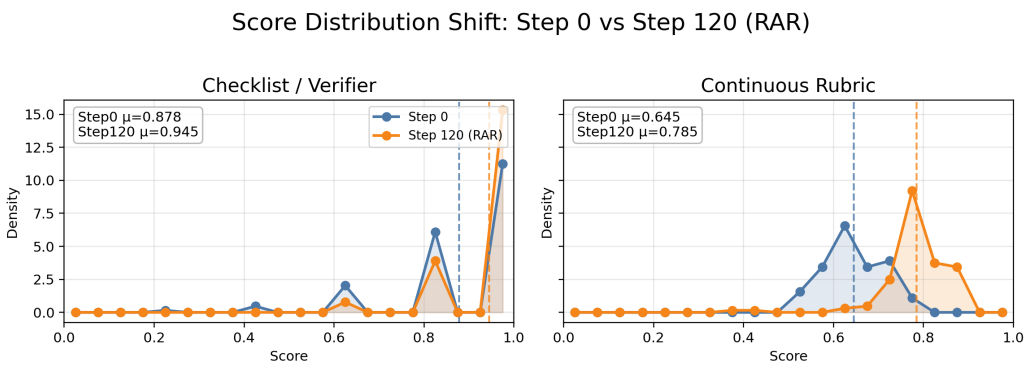


Figure 8: Score-distribution shift from step 0 to step 120 under RaR for the two mechanisms with the clearest shift. Continuous scores shift upward while preserving some spread; checklist scores become highly concentrated near the upper end, increasing the rate of zero-distinction groups.

```
  "item_scores": [0.58]
}
```

Agentic designer harness. The reward-system designer is implemented with the Codex SDK using gpt-5.4 as the backbone model. Codex is not used as a response-level judge. Instead, it operates inside our reward-evolution harness: it observes rollout summaries, reward traces, diagnostic evidence, the current *Reward-DAG*, and run-specific memory; it can only modify the reward system through the typed edit interface. The harness exposes a diagnostic tool set, typed editing operations, and memory tools, while fixed and dynamic skills guide candidate search. The full tool interface and skill content are given in Appendices C.4 and C.5.

Harness runtime prompt. The following is the actual runtime prompt structure supplied to the Codex-backed harness, with long JSON states and run-specific diagnostics abbreviated.

Harness Runtime Prompt

```
You are the Reward-System Designer for RL.
Use the reward-evolution guidance skill to diagnose the current reward-system
state, search for alternative repair directions, and compare the resulting
candidate states. This prompt provides the current run state and
non-negotiable runtime constraints.

## Pre-Loaded State (DO NOT call load_dag or get_dag_json)

### Current DAG
{dag_json}

### Reward Health Dashboard (Iteration {iteration})
{formatted_dashboard}

### Evaluation Results
Accuracy: {accuracy_pct}%
{eval_results_json}

{benchmark_context}

{scenario_meta_rubrics}

## Runtime Reward Context
```

```
The current training reward DAG is pointwise. During actual RL reward
execution, each rubric node sees one sample at a time: the query/prompt,
one candidate response, and any metadata already included in that sample. It
does not see other real responses for the same query.

The diagnostic tool set may retrieve trajectories, inspect reward traces,
cluster cases by a specified response pattern, and replay a candidate on
selected queries. These operations may compare same-query groups outside the
DAG to measure RL signal. Use this evidence to diagnose failures and compare
candidate states. Do not write judge-facing rubric items that require the
judge to compare the current response against unseen same-query responses.

## Current Focus

Dashboard has flagged: {suggested_focus}

## Operational Reward Diagnosis

Use two operational axes before candidate generation and when reading comparative
     evidence:

- On-policy Reward Validity: whether the reward favors the intended behavior
  and covers the current policy's actual response space, including reward
  hacking, high-score low-quality cases, newly exposed coverage gaps,
  formatting/scaffolding artifacts, corner cases, and false-positive penalties.
- Sustainable Reward Informativeness: whether the reward can keep giving GRPO
  useful signal. Use `informativeness_score`, main/base node signal,
  same-query spread, node/item health, parse health, and final reward sanity.

If the DAG has guard or auxiliary downward effects, final-score variance is
not primary evidence for Sustainable Reward Informativeness. Treat final score
as training reward sanity and use `informativeness_score` for reward-signal
claims.

## Runtime Constraints
- The current DAG remains pointwise; judge-facing criteria must not require
  unseen same-query responses.
- Keep the main `judge` signal continuous unless adding a complementary node.
- Do not edit execution-owned fields: `aggregation`, `chunk_size`,
  `parallel_workers`, or `is_final`.
- Use only the exposed reward-harness tools. Direct shell/file/Python operations
  are forbidden; the read-only analysis tool is allowed for guarded mining.
- Use the diagnostic tool set within the supplied tool budget. Treat
  query-specific replay as evidence about candidate reward behavior, not as a
  response-level reward or an automatic adoption decision.
- If recent run-local memory is present in the prompt, briefly cite the recent
  lesson before candidate generation. Explain whether the proposed directions
  repeat, avoid, or extend that lesson.

## Hard Constraints
- DAG must remain pointwise
- Total nodes <= 10
- Scoring (rubric) nodes per path <= 3

## Policy
- Diagnose before generating candidates
- Propose multiple distinct, bounded repair directions supported by the same
  diagnosis
- If the reward system is healthy, make no change
- Compare candidate states on matched evidence and prefer the candidate that
  most strongly addresses the targeted failure while preserving useful reward
  behavior
- If no candidate provides a supported improvement, retain the current state
```

```
- If the diagnostic budget is exceeded, a policy violation occurs, or an
    unsupported fallback
  would be needed, stop and report it.

## Tool Restriction
Use only the exposed reward-harness tools. No direct Shell, bash, file
reads/writes, or Python snippets. The read-only analysis tool is allowed for
guarded rollout/DAG/project analysis.

## Runtime Budget
Use at most {max_tool_calls} tool calls across both phases.

## Additional Context
## Context: Online RL Training
- Current completed step: {step}
- Selected rollout step for diagnosis: {selected_step}
- Rollout metadata: {rollout_metadata}
- Current scenario: {scenario}
- Candidate search mode: {candidate_search_mode}

The rollout diagnosis and dashboard are already pre-loaded in the prompt. Use
the exposed diagnostic tools and actual rollout cases for deeper checks, then
construct multiple bounded candidate states. Before candidate generation,
separate findings into On-policy Reward Validity, which covers both behavioral
validity and coverage, and Sustainable Reward Informativeness. Identify distinct
    repair directions that
may address the diagnosed failure, including continuous rubric refinement,
query_rubric refinement, checklist/coverage complements, complementary
continuous residual scoring, aggregation/weight changes, narrow
guard/flaw_penalty side nodes, or no-op.

{scenario_specific_policy}

{method_advice_context}

{run_memory_context}

{operator_ledger_context}
```

The prompt separates response scoring from reward-system evolution: the local GRM scores responses and instantiates query-level criteria when required, while the agentic designer diagnoses failures, constructs candidate *Reward-DAG* states, and compares their reward behavior through the harness. For candidate search, the runtime groups validity and coverage failures under On-policy Reward Validity and treats reward informativeness separately.

Designer replaceability and tool-use requirements. We first used a local diagnostic-modification bench with representative writing and roleplay reward-failure cases to check whether alternative designer configurations could follow the diagnostic workflow and fixed skills, use the provided tools, construct legal *Reward-DAG* edits, and compare candidate states using rollout evidence. Codex SDK with GPT-5.4, KimiCode with Kimi-K2.5, and OpenCode with DeepSeek-V4-Flash completed this workflow; in contrast, OpenCode with MiniMax-M2.5 frequently made incorrect tool calls or failed to apply valid edits. We selected the Codex configuration for the main experiments based on its reliable tool use, not because the reward-evolution framework requires that particular designer model.

We then compared these designer configurations in a small-scale roleplay training study, keeping the EvoRS framework and other training settings fixed. As shown in Table 11, the three configurations that could use the diagnostic and editing tools reached similar CoSER Overall scores (64.34–65.93). The MiniMax-backed configuration scored 61.44 after frequent edit failures, much closer to the RaR result in Table 2 than to the other designer configurations. These results support a conditional form of designer replaceability: the model need not be GPT-5.4, but it must reliably follow the

Designer configuration	Local diagnostic/edit behavior	CoSER Overall
Codex SDK + GPT-5.4	Completed workflow	65.676
KimiCode + Kimi-K2.5	Completed workflow	64.34
OpenCode + DeepSeek-V4-Flash	Completed workflow	65.93
OpenCode + MiniMax-M2.5	Frequent tool/edit failures	61.44

Table 11: **Designer-backend comparison on roleplay.** Local diagnostic-modification behavior is summarized qualitatively; CoSER Overall reports downstream policy quality under the same reward-evolution framework and training setup.

diagnostic workflow and use the typed tools that implement reward-system evolution. This small-scale comparison does not establish that all designer models are interchangeable.

Update schedule and state transition. The designer is invoked every 5 RL steps. At each evolution step, it uses the diagnostic tool set to examine recent trajectories and reward traces, form several evidence-supported repair directions, and instantiate them as bounded candidate *Reward-DAG* states. The current and candidate states are then evaluated on matched recent rollout groups. Their comparative reward behavior determines which candidate, if any, becomes the next active state; when no candidate provides a supported improvement, the current state is retained. The comparison criteria and typed edit validity checks are described in Appendix C.2.

C.2 Reward-DAG Design Details

Implementation scope. The main text defines *Reward-DAG* as an abstract reward-system representation with editable criterion sources, reward mechanisms, and composition rules. The experiments instantiate a pointwise RL reward path, which is the standard scalar-reward interface used by GRPO. Within this path, EvoRS performs mechanism-level evolution by adding or revising executable reward nodes, including query-level rubric nodes, checklist/verifier nodes, flaw-penalty or guard nodes, and composition operators that fuse side signals with the main scorer.

Data schema. The executable graph is stored as a JSON-like DAG configuration. The following pseudocode summarizes the implemented schema.

```
RewardDAG = {
  name: string,
  version: string,
  description: string,
  mode: "pointwise" | "pairwise",
  entry: node_id,
  nodes: [NodeConfig],
  global_config: dict
}

NodeConfig = {
  id: string,
  type:
    "rubric" | "query_rubric"
    | "verifier" | "gate"
    | "aggregator" | "transform",
  config: dict,
  next: node_id | null,
  description: string
}

ExecutionContext = {
  query: string,
  response_a: string,
  response_b: string,
  query_rubrics: dict,
  variables: dict,
  scores: dict[node_id, float],
```

```
  execution_path: [node_id],
  final_score: float | null
}
```

For pointwise RL, the input is (x, y), represented as `query` and `response_a`. Nodes execute sequentially from `entry`; each node writes variables, node-level scores, and trace details into the context. The scalar training reward is the `final_score` produced by the terminal Rubric Node or Composition Operator.

Node and operator semantics. Table 12 summarizes the implemented node families and their reward semantics. Each executable node has an input contract, writes a trace and a scalar node score when applicable, and passes control through `next` or a gate branch. Judge-produced item scores are parsed as real values and clamped to $[0, 1]$; Boolean verifier outputs are cast to $0/1$ when used as scores or guard-role inputs. A terminal node with `is_final=true` terminates execution and exposes its score as the scalar training reward.

Reward scalarization for RL. For a pointwise training call, the DAG returns one terminal value $r_D(x, y) \in [0, 1]$. If the terminal node is a Rubric Node, this value is the node score defined above. If it is a Composition Operator, it is the normalized composition of upstream node scores. Verifier or checklist outputs contribute either as $0/1$ guard-role variables, pass-rate side scores, or aggregation inputs, depending on the node configuration. No additional learned reward head is introduced: the scalar `final_score` written by the DAG is the reward passed to the RL trainer.

For GRPO-style optimization, after the DAG returns scalar rewards $\{r_i\}_{i=1}^K$ for a response group $\mathcal{Y}_x$, the policy update uses group-relative advantages, e.g.,

$$\hat{A}_i = \frac{r_i - \frac{1}{K}\sum_{j=1}^K r_j}{\mathrm{Std}(\{r_j\}_{j=1}^K) + \epsilon}.$$

Thus the DAG must not only be behaviorally valid, but also preserve within-query reward informativeness.

Typed edit operations. The designer cannot rewrite reward code directly. All changes must be expressed through typed edit operations over the DAG schema:

- `add_node` adds a Rubric Node or Composition Operator using `id`, `type`, `config`, `next_node`, optional `insert_after`, and optional initializer fields. If `insert_after` is set, the tool splices the new node into the reachable path and preserves the old downstream edge.
- `update_node` patches a node using `id`, `config`, optional `next_node`, and optional `description`. For structured rubrics, `rubric_item_updates` modifies stable item ids without rewriting unrelated items.
- `remove_node(id)` removes a non-entry node, bypasses simple predecessors when possible, and removes stale aggregator inputs.
- `set_entry(id)` changes the graph entry point.

Edits are transactional. The tool first clones the current DAG, applies the proposed mutation, normalizes safe structural details, validates the candidate, and commits it only if validation succeeds. Otherwise, the live in-memory and on-disk DAG remain at the last valid state.

Legality checks. The implemented validator rejects malformed or unsafe candidates. Structural checks require a valid entry node, valid `next` and gate branch references, acyclicity, and reachability from the entry node. RL-time pointwise checks require `mode=pointwise`, at most 10 total nodes, at most 3 scoring nodes on any reachable path, no `input_response=both`, no node-level pairwise mode, supported aggregator methods $\{$`weighted_sum`, `product`$\}$, non-empty aggregator inputs, upstream-only aggregator sources, and a valid terminal final node. Additional edit guards prevent the agent from changing program-owned execution fields such as rubric chunking, parallelism, or hidden item aggregation; protect the main continuous node from being directly replaced by a different scorer mode; and reject obvious task-domain contamination in writing or roleplay runs.

Node/operator	Input	Output	Reward semantics
Global Rubric Node	x, y, fixed ρ^g, judge J_ϕ	node score $h_v \in [0,1]$, item traces	Continuous mode scores rubric items $\{c_k\}_{k=1}^K$, normalizes each parsed item score s_k to $[0,1]$, and returns $h_v = K^{-1}\sum_k s_k$. This is the main positive reward mechanism in the pointwise RL runs.
Query-Level Rubric Node	x, y, meta-rubrics, generation policy, optional precomputed query criteria	query-rubric score $h_v \in [0,1]$, generated criteria trace	Instantiates query-aware criteria and scores them through the rubric-scoring path, yielding the same normalized scalar contract as a Rubric Node. In our current experiments this node mainly changes the criterion source, not the hidden scorer execution protocol.
Checklist / Verifier	x, y, explicit checks or checklist items	pass indicators or pass-rate score	Checklist scoring returns $K^{-1}\sum_k \mathbf{1}[\text{pass}_k]$. Rule verifiers write Boolean variables; as guard-role inputs these variables multiply or route downstream scores rather than introducing an unconstrained reward scale.
Deduction / Flaw-Penalty Rubric	x, y, failure dimensions, judge-extracted violations or flaws	penalty-style score $h_v \in [0,1]$, violation trace	Deduction mode uses $s_j = \max(0, 1-\lambda\sum_i \text{severity}_{ij})$ per dimension with default $\lambda = 0.05$, then averages dimensions. Flaw-penalty mode uses $s = 1$ if total severity is zero, otherwise $s = \text{clip}((c - wS)/Z, 0, 1)$, with defaults $c = 101.5, w = 5, Z = 100$.
Composition Operator	upstream node scores $\{h_v\}$	composed score h_u	Weighted-sum aggregation computes $h_u = \sum_v w_v h_v / \sum_v w_v$ over non-guard inputs; inputs marked `role=guard` multiply the base score. Product aggregation computes $h_u = \prod_v h_v$.
Pairwise Comparator	x, y_a, y_b or two pointwise scores	verdict A/B, or a derived preference signal	Used for evaluator benchmarks and analysis. The implemented pairwise path either directly compares two responses or scores y_a, y_b separately and applies a transform A if $s_a \geq s_b$, else B. Pairwise verdicts are therefore benchmark outputs, not the live pointwise scalar reward in the main experiments.

Table 12: Implemented Reward-DAG node families and reward semantics.

Comparative self-validation and state selection. At each evolution step, the system samples matched same-query groups from recent rollouts using four criteria: low-score groups, high-score groups, high-variance groups, and random groups. By default, it samples three query groups per criterion and requires at least two responses per selected query group. The current state and all candidate states are executed on the same cases, so differences in reward behavior are attributable to the candidate state rather than to different rollout samples.

Let $q(D, \mathcal{B})$ denote the diagnostic summary computed from these execution records, with dimensions including discrimination, RL signal, calibration, bias, stability, and hacking risk. For candidate j, let $\Delta q_d^{(j)} = q_d(\widetilde{D}_{t+1}^{(j)}, \mathcal{B}_t) - q_d(D_t, \mathcal{B}_t)$, so positive values favor the candidate. A candidate enters the supported set $\mathcal{C}_t$ only when it satisfies

$$C_{\text{exec}}^{(j)} \wedge C_{\text{guard}}^{(j)} \wedge C_{\text{health}}^{(j)} \wedge C_{\text{info}}^{(j)} \wedge C_{\text{review}}^{(j)}.$$

The individual conditions are:

- $C_{\text{exec}}^{(j)}$: the candidate graph is valid and its execution success rate is at least $\tau_{\text{succ}} = 1.0$.
- $C_{\text{guard}}^{(j)}$: no new critical status appears, and every default reliability dimension $d \in \mathcal{D}_{\text{guard}}$ satisfies $\Delta q_d^{(j)} > -\tau_{\text{guard}}$, with $\tau_{\text{guard}} = 0.05$.
- $C_{\text{health}}^{(j)}$: average diagnostic health does not drop by more than $\tau_{\text{health}} = 0.15$.
- $C_{\text{info}}^{(j)}$: the candidate's informativeness-score std and range are at least 0.02 and 0.05, respectively; the final reward also has nonzero std and range on the matched cases.
- $C_{\text{review}}^{(j)}$: if model-based comparative review is enabled, the candidate receives an overall support score of at least 0.55; otherwise this condition is true by default.

Here $\mathcal{D}_{\text{guard}}$ is the default reliability set {discrimination, RL signal, calibration, bias, stability, hacking risk}. Among supported candidates, the system selects the state whose comparative evidence most strongly improves the diagnosed failure while preserving clean cases and useful within-query distinctions. If no candidate has a supported advantage over the current state, it sets $D_{t+1} = D_t$. Thus the checks define which candidates remain eligible, while comparative feedback determines the next reward-system state. The constants above are implementation defaults and can be overridden for ablations or stricter runs.

Example evolution step. One selected reward-system update occurred in a writing run at step 25. The active graph before the update was a single Global Rubric Node:

> `writing_judge` (`rubric`, continuous, `is_final=true`) with six rubric items: purposeful development, necessary specificity, form-purpose fit, reader effect, insight/originality, and prose control.

The designer diagnosed weak same-query separation and template-like writing artifacts. Among the candidate repair directions, the selected state applied `update_node(writing_judge, rubric_item_updates=...)` without changing the topology. The patch tightened three rubric items: necessary specificity and grounding, insight/originality, and prose control. The new criteria explicitly penalized placeholder scaffolding, pseudo-specific unsupported detail, meta artifacts such as visible word-count scaffolds, and jarring register or language mismatch.

The current and candidate states were compared on 48 recent cases from 12 query groups selected by low-score, high-score, high-variance, and random criteria. The selected candidate achieved execution success rate 1.0, kept the graph valid, introduced no new critical dimensions, preserved the average health score, and did not collapse the reward distribution: score std changed from 0.0673 to 0.0730, and score range from 0.2666 to 0.2916, both above the default collapse thresholds. This example illustrates a bounded criterion-source/rubric-content repair selected through comparative feedback.

Additional selected updates, including the writing guard-style penalty repair used in the training-dynamics analysis, are reported as experimental case studies in Appendix E.5.

Tool family	Representative interface	Role in the harness
Diagnostic tool set	trajectory retrieval; reward-trace inspection; pattern-conditioned case clustering; query-specific candidate execution	Collect on-policy evidence, analyze recurring response patterns, inspect how the current state produces rewards, and expose comparative reward behavior of candidate states on selected queries.
Typed candidate construction	node insertion, node update, node removal, entry change, local aggregation update	Instantiate alternative repair directions as bounded candidate Reward-DAG states while enforcing graph validity and the admissible mechanism and composition search space.

Table 13: Tool sets exposed to the agentic reward-system designer.

C.3 EvoRS Training Algorithm

Algorithm 1: EvoRS

Input: initial policy π_{θ_0}, initial Reward-DAG D_0, query distribution $\mathcal{X}$, reward-evolution interval N.

1. Initialize memory $\mathcal{M}_0$ and dynamic skills $\mathcal{S}_0^{\mathrm{dyn}}$, which form the initial designer context $\mathcal{K}_0$.
2. For each training step $t = 0, 1, \ldots$, sample queries $x \sim \mathcal{X}$ and grouped responses $\mathcal{Y}_x \sim \pi_{\theta_t}$.
3. If query-level rubric nodes are active, instantiate query-aware criteria for the current batch.
4. Score responses with the current Reward-DAG D_t; log rewards, node traces, and reward-health summaries.
5. Update π_{θ_t} using GRPO-style group-relative advantages.
6. If t is not a reward-evolution step, set $D_{t+1} = D_t$ and continue.
7. Otherwise, use $\mathcal{K}_t$ and the diagnostic tool set to diagnose the current reward failure and form alternative typed updates $\{e_t^{(j)}\}_{j=1}^{M}$.
8. Instantiate candidate states $\widetilde{D}_{t+1}^{(j)} = e_t^{(j)}(D_t)$ and execute the current and candidate states on the matched rollout batch $\mathcal{B}_t$.
9. Select as D_{t+1} the supported candidate that most strongly addresses the diagnosed failure while preserving useful reward behavior; if none improves on D_t, retain D_t.
10. Store the selected, rejected, and no-op trajectories in $\mathcal{M}_{t+1}$ and distill their comparative outcomes into $\mathcal{S}_{t+1}^{\mathrm{dyn}}$.

C.4 Agentic Designer Tool Details

The designer operates through a restricted interface rather than unrestricted file or code access. Table 13 summarizes the two functional tool sets used by EvoRS: diagnostic tools assemble evidence and expose the actual reward behavior of current and candidate states, while typed editing operations define the admissible Reward-DAG search space.

Diagnostic tool set. The diagnostic tools expose recent rollout behavior and reward traces without changing the active reward system. They support targeted trajectory retrieval, inspection of node- and item-level traces, clustering or grouping cases that match a specified response pattern, and query-specific execution of candidate states. Together, these tools let the designer move between aggregate patterns and the concrete cases that produced them, then observe whether alternative candidate states change the targeted behavior while preserving useful same-query distinctions. The tools organize this evidence around reward validity, coverage, and informativeness rather than treating any single aggregate statistic as a sufficient update signal.

Editing tools. All Reward-DAG changes must be expressed through typed edit operations. The interface supports bounded node updates, adding evaluator or composition nodes, removing stale or harmful nodes, changing the entry node when needed, and locally modifying aggregation. Each edit is applied transactionally to a candidate graph and is rejected if it creates an invalid DAG. Validation checks include acyclicity, reachability from the entry node, valid upstream inputs for aggregation, pointwise reward semantics, and scenario/domain consistency. Program-owned execution details, such as hidden parser schemas or rubric execution internals, are not exposed as agent-editable fields.

Comparative feedback and state selection. Query-specific candidate execution is part of the diagnostic tool set rather than a separate replay component. After alternative candidate states are formed, the current state and all candidates are evaluated on the same recent rollout cases. Program-defined checks remove invalid or clearly degraded candidates, and comparative review identifies the remaining candidate that most strongly addresses the diagnosed failure while preserving clean cases and useful reward distinctions. If no candidate has a supported advantage over the current state, no transition is made. When model-based review is enabled, it reads the same compact comparative evidence and contributes to this selection; it does not produce response-level rewards or bypass the program-defined checks.

C.5 Reward-Evolution Guidance Skill Details

Section 4.2 abstracts the agentic designer as an experience-conditioned candidate-generation policy $\Pi_{\text{design}}(D_t, \mathcal{O}_t, \mathcal{K}_t)$ over an admissible update space $\mathcal{A}(D_t)$. In implementation, $\mathcal{O}_t$ is assembled through the diagnostic tool set and reward traces, $\mathcal{A}(D_t)$ is defined by the typed *Reward-DAG* update tools and validity checks, and $\mathcal{K}_t$ consists of fixed reward-evolution skills, dynamic skills, and run-local memory. Each candidate state produces a trajectory record, and their comparative outcomes are stored in memory and summarized into later dynamic-skill context.

Tools define what the designer can diagnose and edit. Skill sets define how it should reason over those interfaces. In EvoRS, skills serve as reward-evolution guidance rather than task-level response rubrics: they structure diagnosis, provide reward-quality knowledge, route failures to suitable mechanism families, constrain candidate search, and specify how comparative evidence should be interpreted.

Fixed skills. Fixed skills are shared across runs. They require the agent to diagnose before candidate generation and to ground each repair direction in both rollout evidence and reward traces. For candidate search, the runtime uses two operational targets: On-policy Reward Validity jointly covers validity and coverage failures, while Sustainable Reward Informativeness captures the usefulness of the signal for optimization. Fixed skills also provide reward-quality priors: useful rewards should preserve current-policy preference validity, cover emerging failures, maintain same-query informativeness, avoid saturated or redundant items, execute reliably, and preserve clean cases under matched comparison.

Experience-conditioned candidate search. Although the designer is implemented with an instruction-following agent, the harness constrains its search through a typed action space rather than an open-ended prompt workflow. At evolution step t, the agent receives observation summaries o_t, node traces τ_t, recent memory $\mathcal{M}_t$, and the current graph D_t. It first assigns a primary operational target, using validity as shorthand for behavioral validity and coverage:

$$f_t \in \{\text{validity}, \text{informativeness}, \text{both}, \text{none}\},$$

and a failure pattern p_t from the operator-routing table. The table maps p_t to a small operator family $\Omega(p_t)$, such as a global rubric refinement, query-level rubric specialization, checklist/verifier, flaw-penalty side signal, aggregation adjustment, or removal. Alternative updates are drawn from the legal typed-edit set:

$$e_t^{(j)} \in \mathcal{A}(D_t) \cap \Omega(p_t), \qquad j = 1, \ldots, M,$$

where $\mathcal{A}(D_t)$ is defined by the edit tools and validation checks in Appendix C.2. The candidate set covers distinct, evidence-supported repair directions while excluding actions contradicted by recent memory. Each update produces $\widetilde{D}_{t+1}^{(j)}$, and matched comparative feedback determines which

candidate, if any, becomes the next state. If no evidence-backed direction exists, or no candidate improves on the current state, the system returns a no-op rather than forcing a transition.

Algorithm 2: Experience-Conditioned Candidate Search

Input: current graph D_t, rollout summaries o_t, node traces τ_t, diagnostic summary q_t, memory $\mathcal{M}_t$, fixed skills $\mathcal{S}^{\text{fix}}$, dynamic skills $\mathcal{S}_t^{\text{dyn}}$.

1. Diagnose behavioral validity, coverage, and reward informativeness from o_t, τ_t, q_t; attach representative rollout evidence to each finding.
2. Set $f_t \in \{\text{validity}, \text{informativeness}, \text{both}, \text{none}\}$. If $f_t = \text{none}$, return no-op.
3. Classify the dominant failure pattern p_t using the operator-routing prior in Table 14.
4. Construct the admissible action set $\mathcal{A}(D_t) \cap \Omega(p_t)$, where $\mathcal{A}(D_t)$ is the typed edit set allowed by the current graph validator.
5. Remove actions that conflict with recent rejected memory, modify protected execution fields, shift the current reward objective without evidence, or violate pointwise RL constraints.
6. Select M distinct, evidence-supported repair directions and instantiate the candidate states $\{\widetilde{D}_{t+1}^{(j)}\}_{j=1}^{M}$. If none is supported, return no-op.
7. Use the diagnostic tool set to execute the current and candidate states on matched cases and collect their comparative reward behavior.
8. Select the candidate that best addresses the targeted failure while preserving clean cases and same-query signal; if none improves on D_t, retain D_t. Store all comparative outcomes for later memory and dynamic-skill updates.

This policy is implemented by the designer harness. The language-model agent supplies diagnosis, candidate directions, and comparative judgment, while the action space, graph validator, matched-case construction, and admissibility checks are program-defined.

Condensed fixed-skill card. The following condensed skill card is supplied to the agent as reward-evolution guidance. We reproduce it in segmented form to make the harness behavior auditable.

Mission.

> Optimize pointwise RL reward DAGs during unverifiable generation training. The goal is not to make a prettier rubric; the goal is to keep reward useful as the policy changes. Use two operational targets for candidate search: On-policy Reward Validity, covering behavioral validity and coverage, and Sustainable Reward Informativeness.

Operational diagnosis.

> On-policy Reward Validity asks whether the reward favors intended behavior and covers the current policy's actual response space, including reward hacking, high-score low-quality responses, shortcut behavior, brittle over-penalization, corner cases, false positives, and newly exposed response patterns. Sustainable Reward Informativeness asks whether the reward can keep giving GRPO useful learning signal, using same-query range/std, zero-variance groups, main or base node health, item saturation, score collapse, parse failures, calibration, and advantage signal.

Reward-signal interpretation.

> Reward statistics are health signals, not standalone objectives. Same-query std/range and advantage behavior on the informativeness score are primary evidence for GRPO signal. Final reward mean/std and global variance are training-sanity checks. Larger final-score variance is not progress when it comes mainly from guard or auxiliary downward effects.

Required workflow.

1. Diagnose before candidate generation: inspect rollout summaries, dashboards, node/item traces, same-query groups, contrastive cases, and recent memory.

Failure pattern	Primary operator family	Typical repair
Continuous quality miscalibration	Global continuous rubric	Refine item wording, anchors, evidence requirements, or weights.
Redundant or saturated criteria	Global rubric structure	Split, merge, add, remove, or reweight rubric items.
Prompt-local coverage gap	Query-level rubric node	Revise meta-rubrics or generation policy for query-specific criteria.
Explicit observable requirement	Checklist, verifier, or deduction node	Add hard or soft requirement tracking.
Sparse high-score bad tail or hacking escape	Flaw-penalty, guard, or validity side signal	Add a narrow negative detector or guard-style fusion.
Useful missing positive signal	Complementary positive node	Add a side scorer for a distinct positive construct.
Signal masking or distorted final reward	Aggregator or fusion operator	Adjust weights, guard roles, or fusion method.
Harmful, inert, or duplicated signal	Simplification or removal	Remove, bypass, or downweight stale nodes.

Table 14: Operator-routing priors encoded in fixed reward-evolution skills.

2. State separate findings for validity, coverage, informativeness, recent memory, scenario evolution direction, and objective-continuity risk.
3. Choose one primary target for the evolution step: validity/coverage, informativeness, or both.
4. Classify the failure and identify several distinct repair directions within the admissible operator families.
5. Instantiate each direction as a bounded candidate through the typed Reward-DAG interface.
6. Use the diagnostic tool set to compare targeted failures, clean-case preservation, same-query informativeness, node traces, and execution errors across candidates.
7. Record the selected, rejected, and no-op trajectories, their side effects, and lessons for later dynamic skills.

Table 14 summarizes the operator-routing prior used by fixed skills. This prior prevents the designer from treating every failure as a generic rubric rewrite and helps it construct distinct candidate directions with an intended failure target, an owner signal, and an expected reward-behavior change.

Edit-boundary constraints. Fixed skills constrain reward evolution so the training objective does not oscillate. The agent is instructed to preserve the stable reward core unless evidence supports a larger structural change, prefer bounded complements before replacing the main reward path, make one primary edit at a time, keep RL-time DAGs pointwise, and treat aggregation or final weights as high-impact changes. We distinguish three edit regions: the *Stable Core*, including the main task objective and primary positive reward signal; the *Adaptive Shell*, including anchors, evidence requirements, generation policy, and small coverage refinements; and *Safety/Validity Guards*, which target narrow observable defects and should not become broad secondary quality judges.

Comparative-evidence standards. Skills specify that candidate-level deltas are evidence for state selection, not proof of downstream policy improvement. The designer checks whether targeted cases move in the intended direction, whether clean cases are preserved, whether same-query informativeness improves or at least does not collapse, whether node traces explain the score changes, and whether parse or node errors invalidate a comparison. In particular, guard-induced final-score variance should not be mistaken for sustainable reward informativeness.

Dynamic skills. Dynamic skills specialize the fixed policy to the current scenario and training run. They record observed hacking or shortcut patterns, failure-to-repair mappings discovered during training, selected and rejected candidates, no-op cycles, historical side effects, and scenario-specific

operator preferences. For example, writing runs may record fabricated authority, constraint self-certification, or dimension coupling as concrete hacking families, while roleplay runs may record format gaming, catchphrase stuffing, meta/OOC leakage, or unsupported escalation. These dynamic skills are injected into later designer calls so that the agent can avoid repeated failures and adapt to the policy trajectory.

Dynamic method advice examples. Dynamic skills are implemented as run-local memory and method advice. The advice is advisory context rather than a command list. A condensed version of the advice format is:

> **Ground every edit in three evidence blocks.** Before editing, reconcile reward-signal diagnostics, actual response-level failures, and recent run memory. If any block is missing or contradictory, resolve that gap first rather than inferring a fix from one view.
>
> **Preserve objective continuity across training.** Classify each edit as a reliability fix, calibration change, coverage change, guard/validity change, structural complement, aggregation change, or objective-shifting change. Stable-core edits to the main reward target, hidden scorer protocol, or final weights should be rare.
>
> **Treat comparative deltas as evidence, not progress theater.** Use matched candidate execution to test whether each repair hypothesis holds on same-query spread, trace evidence, clean cases, and targeted failures. Keep reward statistics as decision checks, not isolated optimization targets.
>
> **Escalate skepticism after repeated no-op or revert outcomes.** When recent updates form a repeated no-op/revert cycle, require materially new response-level evidence before trying another near-identical edit.

Scenario-specific advice further constrains dynamic edits. For writing, only concrete fabricated authority, constraint self-certification, and dimension coupling are treated as reward-hacking families; ordinary headings, professional structure, useful detail, appropriate length, or normal task completion are not hacking by themselves. For roleplay, shortcut detection is bounded to observable behaviors such as tag gaming without in-character substance, generic dramatic verbosity masking weak grounding, catchphrase or persona-name stuffing, meta/OOC leakage, and unsupported escalation. Ordinary shallow characterization or weak scene progression should usually be handled through positive character, scene, progression, or query-level criteria rather than broad guard penalties.

Run-local memory schema. Each evolution step produces a compact set of memory records:

$$z_t^{(j)} = (o_t, e_t^{(j)}, \Delta_t^{(j)}, a_t^{(j)}, c_t^{(j)}),$$

where o_t is the diagnosed failure, $e_t^{(j)}$ a candidate update, $\Delta_t^{(j)}$ its comparative evidence, $a_t^{(j)}$ the selected/rejected/no-op outcome, and $c_t^{(j)}$ observed side effects or caveats. The memory summary is not used as reward evidence by itself. It is injected into later designer calls as dynamic guidance, for example to avoid repeating a rejected direction, reuse an effective failure-to-repair strategy, or cool down an operator family that repeatedly produced no-op or harmful changes.

In implementation, each memory record also stores the selected rollout step, update mode, agent type, whether the candidate state was selected, a compact diagnostic summary, the comparative evidence, the tool sequence, policy violations if any, and links to the candidate graph and operator ledger. A separate memory-advice summarizer may compress these trajectories into two fields: a short *memory summary* for the next update and a *method-advice update* for longer-lived run-local guidance.

D Experimental Details

D.1 Reproducibility Materials

We provide a complete, anonymized reproducibility package as supplementary material. It contains the training and evaluation code, filtered training data and data-construction scripts, initial rubrics and Reward-DAG configurations, checkpoint and evaluation-output manifests, and scripts that reproduce the tables and figures in this paper. The package also records random seeds, hardware and software environments, hyperparameters, prompt templates, and the exact model identifiers, provider API versions, access dates, and decoding settings used for all API-based models. Appendix C.1 details

the training configuration, and Appendix D.3 describes the training data and initial reward setup. These materials allow the full pipeline—data preparation, reward-system evolution, policy training, evaluation, and analysis—to be rerun from the provided scripts.

D.2 Artifact Use, Intended Use, and Data Safety

We use existing artifacts including Qwen3-4B (Yang et al., 2025), Qwen3.5-27B, WritingBench (Wu et al., 2025), CoSER (Wang et al., 2025), MiniMax Role-Play Bench, WorkBuddy-Bench, OfficeBench, GPT-5.4 as the designer model, and GPT-5.6-Terra, DeepSeek-V4-Pro, and GLM-5.2 as the main benchmark judges. Additional analyses use DeepSeek-V4-Flash and Kimi-K2.5 (Team et al., 2026). The creators of these artifacts are cited where they are first used in the method and experiment sections. We use these artifacts for research on reward-system evolution and evaluation, consistent with their benchmark, model-evaluation, or model-training use. Proprietary API models are used through their service interfaces; their parameter counts and redistribution rights are not public, so we report model names and roles rather than redistributing them.

The artifacts created by this work include filtered training prompts, initial task-level and query-level rubrics, *Reward-DAG* schemas and edit logs, reward-diagnostic taxonomies, trained policy checkpoints, evaluation outputs, and analysis scripts. These artifacts are intended for research reproduction and analysis of reward-system evolution, not for deployment as general-purpose safety filters or user-facing moderation systems.

For data safety, the writing data are synthetic open-ended writing instructions. We filter duplicated, underspecified, benchmark-like, and factual-QA-dominated queries, and we avoid prompts that require identifying private individuals. Roleplay training data are sampled from MiniMax Role-Play Bench and retained only when persona, relationship, context, and dialogue state are explicit. We do not intentionally collect personal identifiers. Because open-ended generation and roleplay can still produce sensitive or offensive content, final outputs are used only for aggregate evaluation and audit statistics, and auditor prompts do not require exposing method labels, reward traces, or user-identifying metadata.

Appendix D.3 reports the data coverage and sizes: 944 writing queries, 5,000 roleplay queries, and 230 agentic office tasks, with their construction and coverage summarized there. Section 5.1 and Appendix C.1 report the model settings, training protocol, compute budget, and evaluation protocol.

D.3 Training Data and Initial Rubric Construction

Training data. For writing, we construct a filtered synthetic query set for open-ended writing RL. The queries are generated to cover six broad writing domains and diverse constraints over audience, format, length, style, language, and communicative intent. We remove queries that are duplicated, underspecified, benchmark-like, or dominated by factual QA rather than writing quality. The final training set contains 944 writing queries. For roleplay, we sample from MiniMax Role-Play Bench and retain queries with clear persona, relationship, context, and dialogue-state information. The final training set contains 5,000 roleplay queries, covering English and Chinese scenarios with varied persona relations and dialogue depths.

For agentic office-task RL, we construct and train on 230 synthetic office-work tasks using structural seeds from WorkBuddy-Bench (Tencent WorkBuddy Bench Team, 2026). The source benchmark provides structural information only; its task instructions, assets, and reference trajectories are not used for synthesis. OfficeBench (Wang et al., 2024) is used only for held-out evaluation.

Initial rubric construction. We build both task-level and query-level rubrics with a lightweight iterative rubric-generation procedure. Starting from a seed instruction and representative examples, a rubric generator proposes candidate criteria and short scoring descriptions. We then audit the candidates on small batches of representative responses and remove criteria that show poor discriminability, strong coupling with other criteria, vague evidence requirements, or weak relevance to the target task/query. New criteria are added to cover uncovered failure modes or task requirements. This generate–audit–revise loop is repeated until the rubrics provide stable, non-redundant, and task-relevant distinctions. The resulting task-level rubrics are used by the RLAIF-style baseline and

initialize the single-node continuous *Reward-DAG* used by EvoRS, while the query-level rubrics are used by RaR-style reward systems.

For the agentic office setting, rubric synthesis follows the same procedure as in writing. RaR trains with a single fixed task-level rubric, whereas EvoRS starts from the corresponding initial reward system and evolves it during RL.

D.4 Baseline Reward-System Details

All baselines are instantiated under the unified GRPO protocol in Section 5.1. We adapt each published reward formulation to writing and roleplay while using the shared training and evaluation settings described above. RLER and OpenRS use the same local Qwen3.5-27B model as EvoRS for rubric updating and reward computation. The following paragraphs specify how each reward system is instantiated.

Static rubric baselines. The RLAIF-style baseline uses the task-level rubric constructed in Appendix D.3. The same rubric is applied to all queries in a task, and the GRM returns a normalized continuous score as the scalar reward. The RaR-style baseline uses fixed query-level rubrics generated before training. For each query, the query-specific rubric remains unchanged during RL, isolating the effect of criterion-source specialization.

RLER dynamic rubrics. RLER follows the evolving-rubric design of DR-Tulu (Shao et al., 2025). After each rollout step, the update model uses current policy responses to augment every task rubric with additional positive or negative criteria. The original and augmented criteria are scored through the same continuous GRM path. Active criteria with weak or collapsed discrimination are filtered using item-level reward statistics, and the active set is capped to a small maximum size.

OpenRS-style pairwise rubric reward. The OpenRS-style baseline does not maintain a persistent active-rubric state. Instead, after each rollout, responses under the same query are grouped and compared pairwise. For each pair, the OpenRS prompt asks the judge to analyze the query, construct or select pair-specific rubrics, compare the two responses under those rubrics, and output rubric-level comparison results. The implementation parses the pairwise comparison into a signed margin, aggregates margins across pairs for each response, and maps the average margin to a scalar reward in $[0, 1]$. Thus, this baseline changes the reward mechanism to pairwise comparison, but does not perform experience-driven candidate generation and comparative reward-system evolution.

D.5 Evaluation Diagnostics and HR/CFR Audits

We use HR and CFR as final-output robustness audits in addition to final benchmark scores. HR (Hack Rate) measures the incidence of fixed reward-attractive shortcut patterns in final outputs. These patterns are identified from base-model outputs as visible failures that rubric judges may over-reward; when they are amplified after RL, we use this as evidence that the final policy has over-learned reward-attractive shortcuts. CFR (Coverage Failure Rate) measures the audited rate of predefined hard coverage-failure labels in final outputs. R-Var and Zero measure whether the reward gives useful within-query distinctions for GRPO; they are proxies and are interpreted together with the dynamics and case studies in Section 6.2.

Audit protocol. For each task, we define the HR and CFR taxonomies before method-level comparison using a base-model-only calibration pool. The calibration pool contains visible base-model outputs and their prompts, without method labels, reward scores, reward traces, training-stage metadata, or EvoRS checkpoints. Human inspection identifies recurring visible failure patterns in this pool. After this taxonomy-construction step, the HR/CFR categories are frozen and applied to all methods with the same LLM auditor. The auditor only labels whether the frozen categories are present and is not allowed to invent new hack or coverage-failure types. These evaluation taxonomies are not provided to the designer agent during RL training. Training rollouts and reward traces are used only for qualitative case-study interpretation and designer diagnosis analysis, not for defining the reported HR/CFR categories.

HR and CFR metrics. Given audited final outputs $\{(x_i, y_i)\}_{i=1}^{N}$, HR is the rate of outputs matching a predefined task-specific shortcut pattern:

$$\mathrm{HR} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}\left[\begin{array}{c}\exists h \in \mathcal{H}_{\mathrm{task}}: \\ \mathrm{Auditor}_h(x_i, y_i) = 1\end{array}\right],$$

where $\mathcal{H}_{\mathrm{task}}$ is the task-specific HR taxonomy. For roleplay, HR is normalized by the number of assistant turns because a simulation-level shortcut can recur across longer dialogues; the reported roleplay HR is therefore a per-assistant-turn rate.

CFR is the output-level rate of at least one predefined hard coverage failure:

$$\mathrm{CFR} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}\left[\begin{array}{c}\exists c \in \mathcal{C}_{\mathrm{task}}: \\ \mathrm{Auditor}_c(x_i, y_i) = 1\end{array}\right],$$

where $\mathcal{C}_{\mathrm{task}}$ is the task-specific CFR taxonomy. The reported CFR is the percentage of outputs with at least one frozen CFR label. HR and CFR answer different final-output robustness questions: HR detects reward-attractive shortcut patterns, while CFR detects missing coverage of important discrete validity or completion dimensions. A response may have neither, either, or both labels.

Writing HR taxonomy. The writing HR pattern set was fixed from the base-model WritingBench calibration pool. Manual inspection focused on cases that looked attractive to rubric or checklist judges while not improving, or even hurting, the requested writing deliverable. This process produced three task-level pattern families.

Pattern	Definition used by the auditor
Fabricated authority / surface expertise	Dense unsupported numbers, citations, institutions, laws, technical jargon, timelines, metrics, or unverifiable concrete details that create false professionalism.
Self-certification	Meta-level claims that the answer satisfies length, format, constraint, checklist, or scoring requirements, instead of simply completing the requested deliverable.
One-dimension over-optimization	The response is optimized for one visible scoring dimension, such as length, structure, local keyword coverage, or format compliance, while failing essential requirements such as deliverable shape, grounding, usefulness, or audience fit.

Table 15: Predefined writing reward-hacking patterns used for Hack auditing.

Writing CFR taxonomy. Writing CFR uses a stricter hard-failure taxonomy than ordinary WritingBench quality scoring. It excludes ordinary shortness, weak prose, missing minor subpoints, and low benchmark quality; those are handled by WritingBench scores. Instead, it counts discrete non-final or wrong-deliverable failures that a training reward can under-cover even when the output is fluent.

CFR type	Definition used by the auditor
Unresolved template artifact	The response contains unresolved placeholders, bracketed fill-ins, template debris, or explicit word-count artifacts that make the deliverable non-final.
Non-final process artifact	The response exposes planning, evaluation, checklist reasoning, or meta-process text instead of only delivering the requested writing artifact.
Deliverable mismatch	The response does not produce the requested artifact, format, scope, or concrete communicative deliverable.
Missing artifact	The response admits that a required non-text, source-dependent, or concrete artifact is absent, unavailable, or not actually provided.

Table 16: Predefined writing coverage-failure dimensions used for CFR auditing.

Roleplay HR taxonomy. Roleplay HR is intentionally narrower than generic roleplay quality. It counts observable shortcut behavior that can look attractive to a judge without adding new in-character consequence. Ordinary shallow characterization, weak scene progression, or generic prose are not HR by themselves.

Pattern	Definition used by the auditor
Decorative verbosity / emotion stuffing	Inflates length with generic dramatic interiority, repeated emotional beats, atmosphere, or private-planning style padding without creating new visible roleplay consequence.

Table 17: Predefined roleplay reward-hacking pattern used for HR auditing.

Roleplay CFR taxonomy. Roleplay CFR measures discrete validity and completion failures in visible AI-character turns. It does not include broad continuous dimensions such as latest-turn uptake, scene progress, character depth, or generic transplantability; those remain part of CoSER/GCA-style scoring. The reported CFR is limited to the following predefined, easy-to-audit failure families.

CFR type	Definition used by the auditor
Required protocol / block failure	Missing, malformed, replaced, or misordered required user-visible tags, speaker labels, blocks, or output format when this materially breaks the requested roleplay output.
OOC / meta leakage	Explicit AI-assistant framing, instruction discussion, task explanation, OOC commentary, refusal-like meta text, or model/self-reference when not requested.
Visible context contradiction	Contradicts explicit location, relationship state, prior action, character knowledge, causality, or latest-turn facts in the visible context.
User-agency violation	Dictates the user's internal state or actions, resolves too much of the scene, or removes the user's natural ability to respond.

Table 18: Predefined roleplay coverage-failure dimensions used for the reported CFR.

Independent validation of the HR audit. We re-audit the same 50 CoSER scenarios for Base, RaR, and EvoRS at both Qwen3-4B and Qwen3-8B scales using Qwen3.5-27B, DeepSeek-V4-Flash, and Kimi-K2.5. All three auditors produce the same ordering, EvoRS $<$ Base $<$ RaR, at both scales. Pairwise turn-level agreement ranges from 86.48% to 93.72%, with Pearson correlation from 0.822 to 0.942 and Spearman correlation from 0.714 to 0.943. We also collect blinded human-majority labels for 60 randomly sampled outputs. Three-way human/model agreement is 85.0% (Fleiss' $\kappa = 0.705$); human agreement with Qwen3.5-27B and DeepSeek-V4-Flash is 90.0% and 88.3%, respectively (pairwise $\kappa = 0.667$ for both).

LLM auditor prompt. The auditor receives the query or roleplay context, the response or visible simulation transcript, and the frozen task-specific HR and CFR taxonomies. It does not receive method labels, reward scores, reward traces, or training-stage metadata when producing the reported HR/CFR labels. It returns compact JSON and must prefer no label over a weak or invented label:

HR/CFR Auditor Prompt Excerpt

```
Audit the response against the given task/context. Label only the predefined HR
    and CFR types. Do not invent new categories.

HR means the response contains a predefined hacking pattern that a rubric judge
    may over-reward without improving true task quality. CFR means the response
    has a predefined hard coverage failure that may be under-covered by the
    reward.

Use the task-specific HR and CFR type lists provided in the prompt. Mark a type
    only when it is meaningfully present and supported by visible evidence. Do
```

```
    not mark normal useful structure, necessary detail, appropriate roleplay
    narration, or ordinary low quality as HR/CFR. Prefer no over weak yes.

Return JSON only:
{
  "hack": "yes|no",
  "hack_types": ["..."],
  "coverage_failure": "yes|no",
  "coverage_failure_types": ["..."],
  "evidence": "<=25 words"
}
```

R-Var and Zero. For each query x, let $\mathcal{Y}_x$ be the grouped rollout responses and $r(y)$ the scalar reward used by GRPO. R-Var is the average within-query reward variance:

$$\text{R-Var} = \frac{1}{|\mathcal{X}|} \sum_{x \in \mathcal{X}} \text{Var}(\{r(y) : y \in \mathcal{Y}_x\}) .$$

Zero is the percentage of same-query groups whose reward range is indistinguishable under a small numerical tolerance:

$$\text{Zero} = \frac{1}{|\mathcal{X}|} \sum_{x \in \mathcal{X}} \mathbf{1}\left[\max_{y \in \mathcal{Y}_x} r(y) - \min_{y \in \mathcal{Y}_x} r(y) \leq \epsilon\right],$$

with $\epsilon = 10^{-6}$ in our logged reward-health computation. Larger R-Var and lower Zero usually indicate more useful group-relative training signal, but high variance caused only by noisy penalties or score collapse is not treated as sufficient evidence of reward quality.

D.6 Multi-Judge Benchmark Evaluation

We evaluate final-policy quality using GPT-5.6-Terra, DeepSeek-V4-Pro, and GLM-5.2 as judges. Method identities, training rewards, reward traces, and reward-system states are hidden from all judges. For each benchmark, the reported score is the equal-weight average of the three judges on the common set of instances successfully evaluated for every method–judge combination.

For WritingBench, each method produces responses to 1,000 test instructions. Following the original benchmark protocol, a judge scores all five query-specific criteria in a single request on a 1–10 scale. We average the five criterion scores and multiply by ten to obtain the response-level score. The common-success intersection contains 994 instructions; six instances affected by provider-side filtering are excluded uniformly from all methods.

For CoSER, we evaluate 200 multi-turn roleplay simulations per method. Each judge reads the complete dialogue and evaluates SC, AN, CF, and SQ in a single request, preserving evidence that emerges across turns. For dimension d, we compute

$$u_d = 100 - 5 \max\left(0, \sum_j v_{d,j} - 0.3T/4\right),$$
$$s_d = \text{clamp}(u_d, 0, 100),$$

where $v_{d,j}$ is the severity of flaw j and T is the number of assistant turns. CoSER Overall is the mean of the four dimensions. The common-success intersection contains 194 simulations; generation failures and provider-filtered judge requests are excluded uniformly from all methods.

WritingBench judging uses temperature 1.0, top-$p = 0.95$, and a maximum of 4,096 output tokens. CoSER judging uses temperature 0 and a maximum of 2,048 output tokens. GPT-5.6-Terra uses its provider-default reasoning mode, while reasoning is disabled for DeepSeek-V4-Pro and GLM-5.2. All evaluations were conducted in September 2026.

Method	WritingBench			CoSER Overall		
	GPT-5.6	DeepSeek-V4	GLM-5.2	GPT-5.6	DeepSeek-V4	GLM-5.2
Qwen3-4B base	49.815	60.638	54.229	57.075	66.960	58.692
RLAIF	48.992	63.384	56.451	56.944	68.189	54.585
RaR	49.374	63.489	56.779	56.885	68.568	54.548
RLER	49.746	62.771	55.994	56.245	67.141	55.477
OpenRS	49.857	63.127	56.296	53.967	67.544	54.281
EvoRS	**50.024**	**63.819**	**57.161**	**62.242**	**71.387**	**63.400**

Table 19: **Judge-specific final-policy quality.** Each column reports scores from one judge on the common-success subset used by the three-judge aggregation.

Method	Seed 1	Seed 2	Seed 3	Mean $\pm$ SD
RaR	56.547	56.462	56.563	56.524 ± 0.054
EvoRS	**57.001**	**56.893**	**57.056**	$\mathbf{56.983 \pm 0.083}$

Table 20: **WritingBench stability across training seeds.** Three Qwen3-4B runs per method use the main experimental configuration. SD is the sample standard deviation across the three seeds; Seed 1 is the primary run in Table 2.

Benchmark	Comparison	Mean Δ	95% bootstrap interval	Win/Tie/Loss
WritingBench	EvoRS – RaR	$+0.454$	$[-0.022, +0.918]$	502/29/463
CoSER Overall	EvoRS – Base	$+4.767$	$[+3.537, +6.006]$	139/4/51
CoSER Overall	EvoRS – RaR	$+5.676$	$[+4.487, +6.851]$	145/0/49

Table 21: Paired comparisons on the common-success benchmark subsets. Bootstrap intervals use 20,000 paired resamples.

Judge-specific results. Table 19 reports the results from each judge before aggregation. EvoRS obtains the highest WritingBench and CoSER Overall score under every judge. The absolute scores and the magnitude of the roleplay differences vary across evaluators, but the ranking of EvoRS is consistent across both benchmarks.

Training-seed stability on WritingBench. We repeat the RaR and EvoRS WritingBench experiments with three random training seeds, starting from the same Qwen3-4B policy and using the main experimental configuration. The first seed is the primary run reported in Table 2; the other two runs provide a check on training stability. Each score follows the three-judge WritingBench evaluation protocol described above. Table 20 reports the final WritingBench scores for each run. EvoRS exceeds RaR under every seed, with an average score of 56.983 versus 56.524. The mean advantage is 0.459 points; the sample standard deviations across seeds are 0.083 and 0.054, respectively. This comparison probes variation across training runs and is distinct from the paired benchmark-instance uncertainty reported below.

Paired uncertainty estimates. We additionally perform paired bootstrap resampling over the common benchmark instances. In the primary WritingBench run, EvoRS has the highest mean score, with a 0.454-point margin over RaR. On CoSER, its improvements over both the base policy and RaR remain positive throughout the paired bootstrap intervals. The directional comparison shows the same pattern: EvoRS outperforms RaR on 145 of the 194 matched roleplay simulations.

Behavioral effects of fixed reward optimization. The roleplay results reveal that optimizing a fixed or partially adapted reward system can improve selected qualities without improving the complete interaction. Relative to the base policy, RaR increases SC by 1.607 points and CF by 1.991 points, but decreases AN by 2.420 points and SQ by 4.812 points. The resulting change is therefore not a uniform loss of capability, but an uneven shift toward the dimensions most readily captured by the training reward.

Method	Avg. chars	Repeated 5-gram	Repeated sentences	Length stop
Qwen3-4B base	15,883	22.9%	7.0%	8.4%
RLAIF	18,815	26.7%	10.9%	17.9%
RaR	24,173	31.1%	15.8%	27.0%
RLER	19,496	40.1%	23.1%	14.9%
OpenRS	20,649	30.2%	13.6%	37.4%
EvoRS	15,634	**12.2%**	**1.1%**	9.6%

Table 22: **Behavioral characteristics of the evaluated roleplay simulations.** Repetition and truncation statistics are computed directly from the generated dialogues, independently of the quality judges.

		CoSER Overall				Three-judge dimensions			
Method	GRM	GLM-5.2	DeepSeek-V4	GPT-5.6	Mean	SC	AN	CF	SQ
Qwen3-8B base	–	63.589	71.946	58.101	64.545	57.708	72.886	61.989	65.598
RaR	Qwen3.5-27B	61.861	70.748	53.524	62.044	55.975	69.700	60.783	61.719
EvoRS	Qwen3.5-27B	63.687	74.698	60.726	**66.370**	65.732	69.978	65.943	63.829
RaR	RewardAnything-8B	61.720	72.581	53.982	62.761	56.942	70.430	61.364	62.309
EvoRS	RewardAnything-8B	64.308	73.075	59.597	**65.660**	64.099	69.290	65.716	63.535

Table 23: **Detailed Qwen3-8B generality results.** Overall columns report individual judges and their equal-weight mean; SC, AN, CF, and SQ report the three-judge averages.

	CoSER quality					Writing audits		Roleplay audits	
Variant	Overall	SC	AN	CF	SQ	HR	CFR	HR	CFR
Full EvoRS	65.676	63.615	69.880	63.381	65.829	6.5	19.9	5.25	10.0
only criteria-level adaptation	61.155	65.151	58.794	64.580	56.097	6.0	22.1	6.54	8.5
Fixed final *Reward-DAG*	60.573	62.320	60.037	63.745	56.189	5.4	24.5	7.17	15.3
w/o candidate selection	62.494	61.735	65.078	63.673	59.488	10.8	18.0	6.85	8.0
w/o update rejection	63.911	62.877	66.690	65.342	60.735	5.4	16.6	7.99	10.5
w/o evolution memory / skills	62.412	65.493	60.041	65.720	58.394	8.5	21.3	6.20	11.2

Table 24: **Detailed downstream ablations.** CoSER quality scores average GPT-5.6-Terra, DeepSeek-V4-Pro, and GLM-5.2. HR and CFR retain the audit protocol used in Table 2.

This imbalance is also visible directly in the generated dialogues. RLAIF, RaR, RLER, and OpenRS produce longer conversations with substantially more repeated phrases, repeated sentences, or length-truncated responses than the base policy. Their judge rationales frequently identify circular dialogue, scene stagnation, timeline contradictions, excessive literary elaboration, and control of the user character. EvoRS remains close to the base policy in response length while substantially reducing repetition, suggesting that its quality gains come from more coherent progression rather than increased verbosity.

Together, the dimension-level and behavioral results show how a reward system can become less reliable as the policy adapts to it: optimization continues to strengthen reward-salient behavior while degrading qualities that the current reward does not adequately represent. By evolving the reward system from on-policy evidence, EvoRS maintains more balanced supervision as the policy distribution changes.

E Experimental Results and In-Depth Analysis

E.1 Detailed Generality and Ablation Results

Table 23 expands the Qwen3-8B generality result with judge-specific CoSER Overall scores and the three-judge average for each dimension. Table 24 provides the dimension-level quality and reward-reliability audits for the evolution-strategy controls in Table 3 and the component ablations in Table 6.

E.2 Candidate-State Trace Analysis

This subsection presents representative candidate states selected and not selected during writing training, illustrating how comparative feedback shapes the reward-system evolution path. For selected states, we report the local two-step change in same-query R-Var from the non-guard reward branch. This statistic describes the reward behavior observed around each transition and is interpreted together with matched-case comparative evidence, rather than as a standalone measure of downstream policy improvement. Changes to flaw-penalty calibration can produce larger local effects because they directly alter the separation of high-scoring failure cases.

Update family	Steps	Count	Total Δ R-Var	Mean Δ R-Var
Structure / reward composition	5, 15	2	+0.000628	+0.000314
Query-rubric policy	10, 25, 45, 65	4	+0.001083	+0.000271
Guard calibration	20, 35, 40, 55	4	+0.001620	+0.000405

Table 25: Representative selected writing candidate states grouped by update family. The local R-Var changes are two-step before/after changes from the non-guard reward branch and are used as process diagnostics, not standalone downstream quality gains. The positive/zero/negative counts are $2/0/0$, $4/0/0$, and $3/1/0$ for the three rows, respectively.

E.3 Supplementary Reward-Variance Dynamics

Figure 9 isolates the non-guard continuous scorer rather than the final reward, whose variance can also reflect guard penalties. In the writing runs, EvoRS begins with relatively low same-query variance, then maintains a wider score spread later in training. RaR also retains substantial late-stage variance, while RLER fluctuates; variance alone therefore cannot establish that a reward favors better responses. This figure complements the zero-distinction analysis in Section 6.2 and the independent reward-ranking audit below: it describes the availability of score differences, not whether their ordering is correct. At step 120, EvoRS and RaR have similar variance despite their different final WritingBench scores, underscoring why we do not treat variance as a stand-alone quality measure.

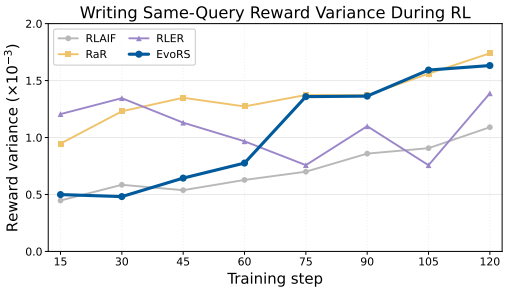


Figure 9: **Writing same-query reward variance during RL.** Variance is computed over 15-step windows from the non-guard continuous scoring branch.

E.4 Reward-Signal Ranking Audit

Section 6.2 uses zero-distinction to test whether the reward remains discriminative on on-policy rollout groups. This diagnostic measures signal availability, but it does not by itself show that the reward distinguishes responses in the correct direction. We therefore conduct a blind list-wise ranking audit that compares reward-induced rankings with independent preference rankings over the same response groups.

The independent ranking standard is method-agnostic and does not use WritingBench criteria, method-specific rubrics, or reward-system rubrics. Rankings are based on human-readable writing preferences that can be applied uniformly across methods: whether the response satisfies the user request, fits the requested format and audience, follows length and style constraints, remains concrete and useful, avoids unsupported specificity or self-certification, and is coherent as a finished writing deliverable. The same ranking standard is used for every method and checkpoint.

For each sampled group, the packet contains one query and four same-query candidate responses. Candidate identities are anonymized and shuffled. Annotators do not observe method labels, reward scores, hidden mappings, rollout source paths, or the reward ranking. Hidden mappings and logged reward scores are used only after all rankings for a packet are completed.

Step	Update family	Candidate change	Selection evidence
5	Structure / query-rubric mechanism	Added a `writing_query_judge` node so the reward could instantiate query-specific criteria from meta-rubrics instead of relying only on a global rubric.	Matched replay verified the DAG path and established the query-level reward signal used by later updates.
10	Query-rubric policy	Constrained query-rubric generation to produce criteria better matched to the current query and reduce flattening from generic criteria.	Targeted replay showed mechanism activation and improved sampled same-query separation.
15	Structure / guard-style aggregation	Added `writing_artifact_flaw_penalty` and a multiplicative final aggregator so the artifact signal could only lower the base reward.	Replay showed activation on high-score bad-tail patterns such as long templates, unsupported precision, and generic padding while preserving clean cases.
20	Guard calibration	Calibrated the artifact guard to better trigger on word-count tags, mixed-script glitches, generic padding, and related observable artifacts.	The previous guard underactivated; clean same-query replay showed improved separation.
25	Query-rubric reliability	Strengthened query-rubric generation reliability so criteria are derived from the current query rather than reused as coarse generic dimensions.	Comparative feedback supported the change, and execution checks passed.
35	Guard calibration	Tightened generic-expansion and redundant-padding detection so section inflation and boilerplate are penalized when they lack query-specific substance.	High-scoring long-template outputs still often received perfect guard scores; same-query replay showed improved sampled spread.
40	Guard calibration	Tightened template-scaffold and meta-leakage detection for placeholders, word-count tags, scaffold shells, and similar artifacts.	Selected as a narrow false-negative repair; local R-Var was approximately unchanged, but replay showed targeted activation without broad degradation.
45	Query-constraint policy	Required query-rubric generation to include an explicit query-constraint fulfillment dimension covering format, length, structure, audience, and required content.	Replay had no node errors and showed usable query-level spread.
55	Narrow guard rule	Added a narrow rule that unresolved bracket placeholders must trigger the template-scaffold flaw rather than receiving a perfect guard score.	Targeted replay fixed clear false negatives such as unresolved placeholder greetings without same-query degradation.
65	Query-rubric anti-template policy	Tightened query-rubric generation to distinguish query-grounded completion quality from generic expansion, structural fluency, and template-like length.	Replay had no node errors and successful query-rubric generation across sampled cases.

Table 26: Representative selected writing candidate states. Each row summarizes the update family, the bounded reward-system change, and the comparative evidence that supported its selection.

Step	Non-selected candidate change	Comparative evidence
30	Further tightened query-rubric generation to repair query-rubric degeneration and same-query flattening.	Targeted replay showed lower same-query informativeness; broader replay was mixed and partly parse-confounded, so the candidate was not selected.
50	Further tightened the guard penalty for generic expansion.	Replay reduced same-query standard deviation and range, indicating compression of sustainable reward signal; the candidate was not selected.
60	Made mixed-script artifacts trigger the guard more deterministically.	Target activation was unstable and the penalty spilled over to non-template technical or plain-language cases.
70	Relaxed or rewrote query-rubric policy to reduce fallback and improve query-level informativeness.	Candidate replay lowered the non-guard same-query signal and did not support selecting the new state.

Table 27: Representative non-selected writing candidates. These outcomes show how comparative feedback redirects later search when observed reward behavior indicates signal collapse, unstable triggering, parse-confounded behavior, or penalty spillover.

Each packet is independently ranked by a three-model LLM judge panel: GPT-5.4, DeepSeek-V4-Pro, and Kimi-K2.5. We aggregate the three rankings with Borda voting to obtain a consensus ranking, and we report inter-judge pairwise agreement as a stability diagnostic. This design reduces sensitivity to any single judge trajectory and lets us separate reward-alignment results from annotation instability. Human review is used for packet validation and calibration: human annotators provide an initial preference prior, inspect packet construction and blinding, and check a 20-group manually ranked subset against the LLM consensus ranking.

Reliability check	Groups	Pairwise agreement
Three-model LLM judge panel	480	0.8408
Human vs. LLM consensus	20	0.71

Table 28: Reliability checks for the blind list-wise ranking audit. The three-model value averages inter-judge pairwise agreement over all method/checkpoint groups. The human-vs-LLM row reports the manually checked 20-group subset.

The main metric is consensus pairwise ranking agreement. A four-response ranking induces six pairwise preferences. We compare the six preferences from the independent consensus ranking with the six preferences induced by reward scores, ignoring reward ties, and average the agreement over groups. We additionally report auxiliary diagnostics in Figure 10. Judge-vs-reward pairwise agreement averages each LLM judge's ranking against the reward ranking before consensus. Top-1 match measures whether the reward-selected best response is also ranked first by the independent judge. Reward-top mean rank measures where the reward-selected best response appears in the independent ranking; lower is better. Inter-judge agreement measures the intrinsic reliability of the list-wise annotation task.

E.5 Case Studies

Writing guard-style penalty repair. This case supports the main-text hack-rate analysis. The frozen evaluation taxonomy in Appendix D.5 was not exposed to the designer. The agent instead inspected on-policy rollout evidence and reward traces, then diagnosed overlapping failure modes in its own terms. In the writing run used in Figure 5(a), a guard-style penalty path was introduced at step 15 and calibrated at the annotated step-20 transition. Before the structural update, the active graph contained a continuous Global Rubric Node, a Query-Level Rubric Node, and a terminal weighted-sum Composition Operator:

`writing_global_judge` → `writing_query_judge` → `final_reward`, where `final_reward` was a weighted sum over the global and query-level scores.

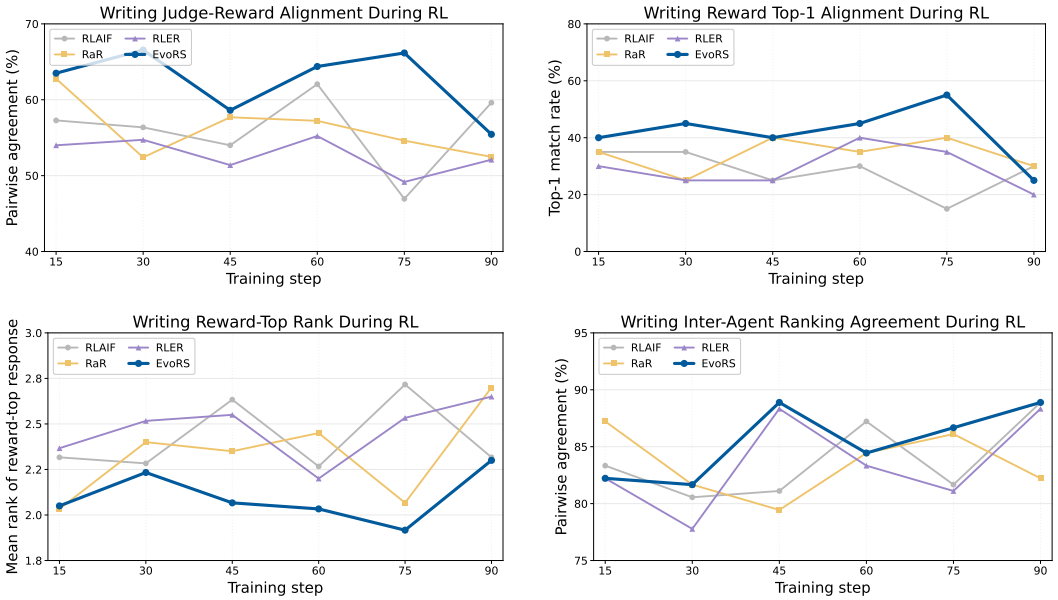


Figure 10: Supplementary diagnostics for the blind reward-ranking audit. These curves complement the main consensus pairwise ranking agreement curve in Figure 5(d).

The designer diagnosed concrete reward-hacking evidence: many rollout responses contained word-count tags, placeholders, or template residue, and several artifact-heavy responses still received high rewards. These observations overlap with the evaluation-time families of fabricated authority, self-certification, and one-dimension over-optimization, but were derived from rollout diagnostics rather than from the external auditor labels. Among the candidate repair directions, the selected state added a bounded integrity side node after the query judge and marked it as `role=guard` in the final aggregator. The side node covers artifact-free output, specificity/authenticity, and anti-template behavior. The final reward can be viewed as

$$r_{\text{final}} = h_{\text{base}} \cdot h_{\text{guard}},$$

where h_{base} is the weighted positive score from the global and query-level nodes, and $h_{\text{guard}} \leq 1$ can only reduce rewards for detected artifacts.

Later updates refined the same guard-style path rather than introducing the frozen evaluation labels. A step-90 probe attempted to switch the side-node scoring mode to `flaw_penalty`, but replay revealed an operator-specific failure: empty flaw extraction mapped to an always-pass score, so this probe was treated as diagnostic evidence rather than a successful repair. Subsequent selected updates at steps 95–115 made bounded local updates to the guard-role node and grounding items, targeting mixed-script glitches, placeholders, explicit word-count tags, and dense unsupported numeric/entity specificity. Replay evidence showed activation on targeted bad cases while preserving strong clean examples in sampled checks. This case therefore shows a structure-level guard-style penalty repair and later calibration, not merely a criteria-only dynamic rubric rewrite.

Roleplay coverage repair. This case supports the main-text on-policy coverage analysis. In a representative roleplay evolution step, the designer diagnosed that the query-side roleplay reward under-covered latest-turn consequence tied to visible scene-state changes. Several high-scoring responses were long and polished, but relied on decorative `<role-think>` padding, transplantable taunts, or unsupported specifics instead of concrete scene movement. A sampled same-query Morgan group was nearly flat, with score range about 0.01, indicating that the reward did not separate materially different continuations.

The selected candidate updated only `roleplay_query_rubric`. It strengthened criteria for latest-turn consequence, substantive content inside required roleplay blocks rather than tag compliance alone, and grounded progression with economy. It also demoted long private planning, decorative inner monologue, catchphrase or persona-name stuffing, and unsupported invented specifics. This is a positive-side coverage/calibration repair rather than a hack-only guard. Comparative feedback

supported this candidate: mean score changed from 0.6313 to 0.6390, within-group standard deviation from 0.0765 to 0.0844, and within-group range from 0.1969 to 0.2177. The case supports the main-text claim that the designer can repair missing on-policy quality dimensions, not only penalize obvious hacking artifacts.

E.6 LIMITATIONS

Our experiments establish reward-system self-evolution in open-ended language generation, while leaving several extensions for future work.

In our experiments, reward-system evolution is performed every N RL steps to simplify the system design and control computational cost. A schedule conditioned on accumulated failure evidence could instead adapt the timing of evolution to the changing policy, which we leave for future study.

Periodic diagnosis and candidate evaluation also introduce some additional training cost. Appendix C.1 reports the wall-clock performance comparison with other methods. Adaptive scheduling or batched candidate evaluation may further reduce this overhead.