RLAR: AN AGENTIC REWARD SYSTEM FOR MULTI-TASK REINFORCEMENT LEARNING ON LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model alignment via reinforcement learning depends critically on reward function quality. However, generic reward models often underperform on heterogeneous task distributions due to distribution shifts, while training task-specific reward models is costly and prone to annotation difficulty, catastrophic forgetting, and loss of generalization. We present RLAR (**R**einforcement **L**earning from **A**gent **R**ewards), an agent-driven framework that dynamically assigns tailored reward functions to individual queries. Specifically, RLAR transforms reward acquisition into a dynamic tool synthesis and invocation task. It leverages LLM agents to autonomously retrieve optimal reward models from the Internet and synthesize code-based verifiers through code generation. This allows the reward system to self-evolve with the shifting data distributions during training. On REWARDBENCH-V2, RLAR outperforms static baselines and approaches the performance upper bound, validating the dynamic reward orchestration mechanism. In online RL post-training, RLAR yields consistent gains of 10%–60% across mathematics, coding, translation, and dialogue. Ablations confirm that synthesized code-based verifiers complement neural reward models on rule-heavy tasks, and that precise per-query reward routing outperforms monolithic reward prediction.

1 INTRODUCTION

Large language model (LLM) alignment via reinforcement learning (RL) depends on reward quality (Schulman et al., 2017; Ouyang et al., 2022; Shao et al., 2024). On heterogeneous tasks, generic reward models often lack domain-specific discriminative power under distribution shifts. Training specialized models is costly and requires expert annotations, while introducing risks of catastrophic forgetting and poor generalization.

Open-source platforms such as *HuggingFace* and *ModelScope*[1] already host numerous task-specific reward models (Liu et al., 2024; Cai et al., 2024; Yang et al., 2024; Corrêa, 2023; Cheng et al., 2025; Lambert et al., 2025). These models often outperform generic evaluators within their domains but remain underutilized. Rather than training another static reward model, we propose orchestrating these existing models to match each query's requirements.

We introduce **RLAR** (Reinforcement Learning from Agentic Rewards), a self-evolving framework that **treats reward functions as callable tools**. An LLM agent selects suitable tools for each query. When the local library lacks a suitable tool, it retrieves a neural reward model from online repositories or generates a code-based verifier, expanding the library as new tasks arrive. We validate reward orchestration on REWARDBENCH-V2 (Malik et al., 2025b) before online RL training. This shifts the agent's role from generating reward scores to **designing and selecting reward functions**.

We evaluate RLAR on mathematics (Hendrycks et al., 2021; Cobbe et al., 2021), code (Xia et al., 2025; Austin et al., 2021), translation (NLLB Team et al., 2022), and general dialogue (Ding et al., 2023), against classification (Liu et al., 2024) and generative (Tu et al., 2024) reward models and GPT-5-as-a-judge. RLAR improves overall performance by 10.4% on Llama-3.1-8B and

[1] https://huggingface.co/, https://modelscope.cn/

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.

61.9% on Qwen3-8B, while resisting **format and verbosity hacking**. Compared with GPT-5-as-a-judge, it reduces API token consumption by approximately 80% and GPU training hours by 75%. Ablations show that **CodeVerify** supports rule-heavy tasks, **WrapLLM** supports open-ended tasks, and random reward assignment degrades nearly all benchmarks, demonstrating the importance of per-query routing beyond tool diversity. Data and code are available at https://anonymous.4open.science/status/ICLR2027-RLAR.

2 PRELIMINARIES

2.1 TASK DEFINITION

We investigate the problem of reinforcement learning in LLM post-training stage for complex, mixed-domain text generation tasks. The core objective is to train a single policy model that achieves high performance across multiple task domains without sacrificing the quality of any individual task. Let $\{D_1, D_2, \ldots, D_n\}$ denote datasets from n different domains, each corresponding to one type of task (e.g., translation, summarization, question answering). We define a mixed-domain distribution $D = \{D_1, D_2, \ldots, D_n\}$. Our goal is to train a policy model A that maximizes expected performance over D under a multi-task reinforcement learning framework:

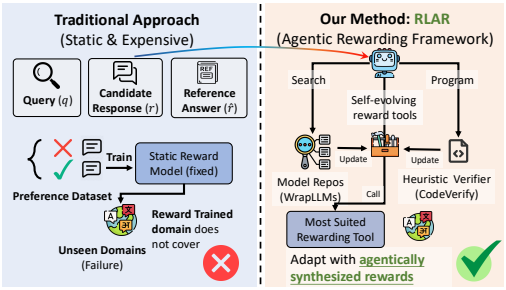


Figure 1: A conceptual comparison between traditional reward modeling and our proposed RLAR framework.

$$\max_{A} \mathbb{E}_{d\sim D}\left[R(A(d))\right],$$

where $R(A(d))$ denotes the reward of policy inference on sample d. The reward is computed from test-time metrics, such as success rate, accuracy, LLM-judge score, and other task-specific metrics. We study the blended domains where real LLM post-training usually focuses: **mathematical reasoning**, **code and programming**, **translation**, and **general conversation** tasks. We show information about the datasets used in our experiments in Table 1.

2.2 LLM POST-TRAINING WITH ONLINE RL

In reinforcement learning from human feedback (RLHF), the **Proximal Policy Optimization** algorithm (Schulman et al., 2017) is frequently employed for policy optimization. The typical workflow begins with a *warm-start training* phase in which a **Value Model** (often a reward model) is learned. Training triplets (x, y_+, y_-) are sampled from human preference data, with y_+ being preferred over y_-. Let $r_\theta(x, y)$ be the reward assigned by the model given prompt x and response y; the learning objective function can be expressed as:

$$\mathcal{L}(\theta) = -\frac{1}{N}\,\mathbb{E}_{(x,y_+,y_-)\sim D}\left[\log\sigma\big(\Delta r_\theta\big)\right],$$

where $\sigma(\cdot)$ is the logistic sigmoid function and $\Delta r_\theta = r_\theta(x, y_+) - r_\theta(x, y_-)$.

We adopt a more training-efficient framework. The **Group Relative Policy Optimization** (GRPO) (Shao et al., 2024) modifies the advantage estimation to reduce the dependence on a learnable value model for estimating the advantage baseline. Instead, GRPO computes the normalized advantage within a group of sampled outputs $\hat{A}_i = \frac{r_i - \mathrm{mean}(r)}{\mathrm{std}(r)}$, where $r = \{r_i\}_{i=1}^{G}$ are the rewards assigned to G candidate outputs for the same prompt, $\mathrm{mean}(r)$ and $\mathrm{std}(r)$ are computed over the group. Also, the Kullback-Leibler Divergence term is removed from the per-step reward and is

Table 1: Summary of domains considered across Math, Code, Translation, and General Conversation.

Domain	Dataset	Train Size	Test Size	OOD	Evaluation Metric
Math	GSM8K	1,500	1,319	✗	Numeric Match
	HENDRYCKS-MATH	1,500	500	✗	
	AIME-24/25	0	60	✓	
Code	LEETCODE-DATASET	2,585	223	✗	Unit Test Execution
	MBPP	0	974	✓	
Translation	FLORES-200	3,000	300	✗	Hybrid (LLM-Judge + BLEU)
	WMT-24	0	500	✓	
General	ULTRACHAT	2,500	200	✗	Model-based (LLM-as-a-Judge)

instead applied directly to the overall optimization objective:

$$\max_{\phi} \mathbb{E}_{\substack{x\sim D,\\ \{y_i\}_{i=1}^{G}\sim\pi_{\text{old}}(y_i|x)}} \left[\frac{1}{G}\sum_{i=1}^{G} \min\left\{ \frac{\pi_\theta(y_i|x)}{\pi_{\text{old}}(y_i|x)}, \text{clip}\left(\frac{\pi_\theta(y_i|x)}{\pi_{\text{old}}(y_i|x)}, 1-\epsilon,\ 1+\epsilon\right) \right\} \hat{A}_i, \right] - \beta\, \mathbb{D}_{\text{KL}}[\pi_\phi \parallel \pi_{\text{ref}}]\,. \tag{1}$$

3 METHODOLOGY

RLAR dynamically synthesizes, verifies, and invokes reward tools for heterogeneous tasks, using an evolving library of neural models and executable verifiers.

3.1 PROBLEM FORMULATION

We formalize the RLAR framework as a dynamic reward synthesis and calling problem. Given an input context triplet $(q, r, \hat{r})$ comprising a query, the candidate response, and the reference answer, our objective is to compute a precise scalar reward $s \in \mathbb{R}$ that reflects the response quality.

The framework operates on a **Reward Toolset (Library)** $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$, where each $f_i : (q, r, \hat{r}) \to s$ is an atomic reward tool (e.g., an LLM-classifier-based reward model, a code-based checker, or a heuristic rule). Initially, $\mathcal{F}$ is populated with a general-purpose base reward model (e.g., Skywork-Llama). As the framework encounters more queries from diverse tasks, $\mathcal{F}$ expands through agentic synthesis: $\mathcal{F}_{t+1} = \mathcal{F}_t \cup \{f_{\text{new}}\}$.

3.2 ADAPTIVE REWARD ROUTING AND SELECTION

A core challenge in RLAR is determining whether the current toolset $\mathcal{F}$ can accurately evaluate a given $(q, r, \hat{r})$. We introduce a **Policy Router** π_{agent}, which serves as the decision-making core.

Completeness Assessment: For each incoming $(q, r, \hat{r})$, π_{agent} assesses the *competency* of the existing library. We formalize this as a classification-of-intent task:

$$a = \pi_{agent}(\text{state} = (q, r, \hat{r}), \mathcal{F}) \tag{2}$$

where $a \in \{\text{Select}, \text{Synthesize}\}$.

Tool Selection: If $\mathcal{F}$ is complete ($a =$ Select), a selection module ϕ_{select} identifies the optimal reward tool $f_t \in \mathcal{F}$ by matching the task metadata with tool descriptions.

Tool Synthesis: If no suitable tool exists ($a =$ Synthesize), it calls the Agentic Synthesis module to construct the missing reward tool.

3.3 AGENTIC TOOL SYNTHESIS

When a tool is missing from the reward tool library $\mathcal{F}$, RLAR invokes specialized agents to construct a new reward tool f_{new}. Considering different reward system types, we design **WrapLLM** and **CodeVerify** to fulfill this goal.

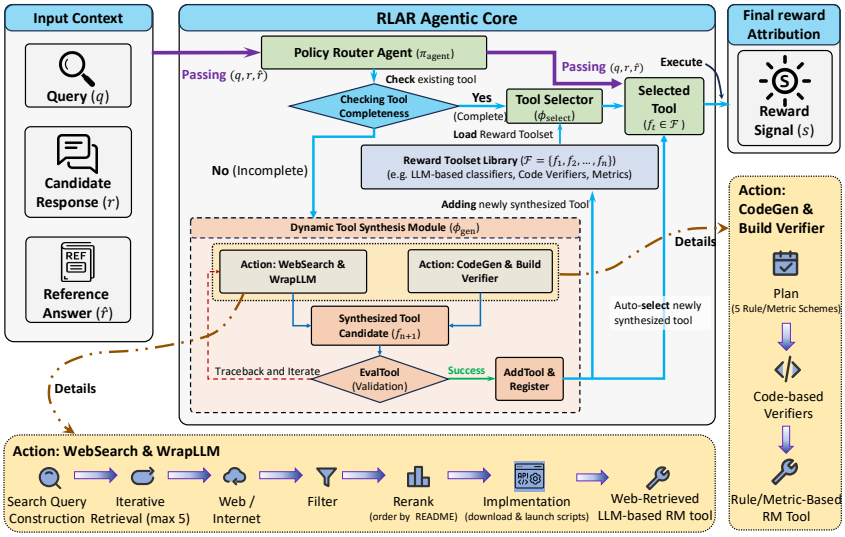


Figure 2: The Reinforcement Learning with Agentic Reward (RLAR) Framework.

3.3.1 WrapLLM

The LLM agent retrieves and encapsulates the most compatible lightweight (within 10B) reward model repository from the open-source model platform *HuggingFace*. These are suited for tasks requiring semantic nuances, creativity, or value alignment. The agent follows an iterative retrieval workflow: (1) Generate a search query for the search engine; (2) Iteratively retrieve from the search engine; (3) Filter and rerank results for the most related reward model; (4) Download and implement the deployment and inference code; (5) Wrap (generating the function names, descriptions) into an API for tool calling. The workflow transforms a reward model checkpoint into a callable tool for $\mathcal{F}$.

3.3.2 CodeVerify

The agent follows a Plan-and-Execute strategy to generate deterministic verifiers for objective tasks such as mathematics and programming. It initially plans up to five possible schemes and implements the optimal one as a reward tool. The agent generates Python-based scripts that vary by task type. For example, retrieving the final answer from a formatted solution is suited for math; retrieving and arranging generated code and executing it with test cases is suited for code. Furthermore, for tasks such as translation, it implements a metric-based (e.g., ROUGE, BLEU, METEOR) reward tool. The prompts for the above are provided in Appendix D.4.

3.4 Verification and Library Update

To ensure the reliability of the synthesized tools, each f_{new} must pass a module (**EvalTool**) before being committed to $\mathcal{F}$. For *WrapLLM*, it examines the consistency between the model-as-a-tool description and its original documentation. For *CodeVerify*, it performs analysis on code implementation and execution checks on synthesized scripts. Only tools that satisfy these verifications are committed to the library: $\mathcal{F} \leftarrow \mathcal{F} \cup \{f_{\text{new}}\}$.

3.5 Evaluating Reward Orchestration and Selection

Before online RL training, we isolate the agent's ability to select a suitable reward function for each query on RewardBench-v2.

Setup. We uniformly sample 400 instances from RewardBench-v2, each containing one preferred and three rejected responses.[2] A reward model passes an instance if the softmax-normalized score of the preferred response exceeds 0.5. We use evaluation records for the leaderboard's top 50

[2]The "Tie" category is excluded due to the test's input-output format uniformity.

Table 2: Comparison of Agentic Selection vs. Reward Merging on REWARDBENCH-V2.

Method	Avg	Precision IF	Math	Safety	Factuality	Focus
SOTA	87.19	57.14	60.00	97.12	85.29	99.13
Agentic Selection within RLAR *framework*						
Random	78.39	40.48	60.00	89.42	75.49	90.43
GPT-4o	88.19	57.14	68.57	96.15	87.25	99.13
GLM-4.6	88.69	57.14	68.57	96.15	88.24	100.0
GPT-5	**90.44**	**66.67**	**71.42**	96.15	**90.19**	**100.0**
Logits Merging Methods						
mean@2	87.44	61.90	68.57	95.19	83.33	99.13
mean@10	77.89	47.62	57.14	84.62	76.47	90.43
mean@50	51.51	40.48	48.57	66.35	48.04	46.09
Theoretical Best	93.47	76.19	77.14	97.12	95.10	100.0

classification models, presented as callable tools. The agent selects a model from the query, candidate responses, and model metadata. We compare **backbone LLM scaling** with GPT-4o, GLM-4.6, and GPT-5, and **logits merging**, which averages the top-k models' predictions (mean@k).

Results. Table 2 shows that averaging reward logits does not expand the performance ceiling and degrades as more models are included. Stronger agentic backbones improve routing accuracy, approaching the available model pool's theoretical upper bound. We next evaluate this mechanism in online RL.

4 EXPERIMENTS

4.1 DATASET AND EVALUATION

We conduct RL training and evaluation across four representative task domains, each chosen to stress-test different aspects of the reward system. **Math Reasoning:** We use GSM8K (Cobbe et al., 2021) and HENDRYCKS-MATH (Hendrycks et al., 2021) for mathematical reasoning. The goal is to reason toward the final numeric solution. **Coding:** We select the LEETCODE-DATASET (Xia et al., 2025), which requires the model to fill in LeetCode cloze-style function implementations. **Translation:** We sample from FLORES-200 (NLLB Team et al., 2022) to cover 20 translation pairs across five languages (CN, DE, EN, FR, JP). This task requires mastery across languages and is important for LLMs' accessibility to worldwide users, especially non-English speakers. **General Alignment:** ULTRACHAT (Ding et al., 2023) is employed to train and evaluate multi-turn dialogue and instruction-following capabilities.

Generalization Challenge. To assess the robustness of the trained policy, we include benchmarks such as AIME-24/25 for math reasoning, MBPP (Austin et al., 2021) for Python synthesis, and WMT-24 for unseen translation contexts. These benchmarks evaluate whether the post-trained model can generalize rather than overfit to the specific heuristics of the training reward models, as an indication of reward hacking.

Data Processing. For all training queries, we apply an automated filtering process, requiring an LLM to discard samples with low query or response quality. The prompt is provided in Appendix B.2. We downsample the training sources—GSM8K, HENDRYCKS-MATH, LEETCODE-DATASET, FLORES-200, and ULTRACHAT—to form the mixture in Table 1. AIME-24/25, MBPP, and WMT-24 are reserved for evaluation. To facilitate our experiments, we obtain the reasoning outputs of GPT-5 on all queries. For dataset queries lacking human-annotated reasoning references, we supplement them with the results from GPT-5. Table 1 shows the statistics of the train and validation sets. A more detailed introduction of the tasks and our selected datasets is provided in Appendix B.

Evaluation Metrics. To ensure evaluation rigor, we apply Numeric Exact Match (**NEM**) at Best@1 for mathematical reasoning tasks. For code generation, we report **pass@1** verified by automated code executors. For translation tasks, we report both **BLEU** (Papineni et al., 2002) and **LLM-judge** scores, while general dialogue tasks are evaluated solely by the LLM-judge. Additionally, we measure the

Table 3: **Main Results (Math & Code)**. **Bold**/underline: best/second-best. ↑/↓: changes vs. Zero-Shot baselines. **NEM**: Numeric Exact Match. HEND: HENDRYCKS-MATH. LCODE: LEETCODE. **Bold** datasets denote unseen training data. **Avg.**: overall average across all four task domains (shared with Table 4). **M&C**: local average over Math & Code benchmarks in this table only.

Domain	**Avg.**	**M&C**	**Math**				**Code**	
Dataset	**Score**	**Avg.**	GSM8k	HEND	**AIME24**	**AIME25**	LCode	**MBPP**
Metric	-	-	NEM	Pass@k	NEM	NEM	Pass@k	Pass@k
Llama-3.1-8B								
Base	37.81	28.23	60.50	49.80	6.67	0.00	11.21	41.17
SkyLlama	23.47	13.17	11.30 ↓49.20	47.60 ↓2.20	3.33 ↓3.34	0.00 -	8.97 ↓2.24	7.80 ↓33.37
SeedX	17.29	10.64	1.21 ↓59.29	42.20 ↓7.60	6.67 -	0.00 -	0.00 ↓11.21	13.76 ↓27.41
SkyQwen	31.68	18.96	16.15 ↓44.35	46.60 ↓3.20	3.33 ↓3.34	0.00 -	**11.66 ↑0.45**	36.04 ↓5.13
SkyCritic	34.05	26.43	44.58 ↓15.92	48.00 ↓1.80	10.00 ↑3.33	0.00 -	10.31 ↓0.90	**45.69 ↑4.52**
GPT-5-Judge	39.89	30.14	71.95 ↑11.45	43.20 ↓6.60	6.67 -	6.67 ↑6.67	7.17 ↓4.04	45.17 ↑4.00
Human	33.56	22.79	42.38 ↓18.12	40.00 ↓9.80	3.33 ↓3.34	3.33 ↑3.33	10.31 ↓0.90	37.37 ↓3.80
RLAR(Ours)	**41.73**	**33.84**	**73.61 ↑13.11**	**54.60 ↑4.80**	**13.33 ↑6.66**	**6.67 ↑6.67**	**11.66 ↑0.45**	43.18 ↑2.01
Qwen3-8B								
Base	29.12	12.61	31.99	19.80	3.33	0.00	6.28	14.27
SkyLlama	29.25	12.49	0.00 ↓31.99	35.80 ↑16.00	10.00 ↑6.67	6.67 ↑6.67	7.62 ↑1.34	14.85 ↑0.58
SeedX	35.14	21.26	0.00 ↓31.99	**72.60 ↑52.80**	6.67 ↑3.34	6.67 ↑6.67	21.08 ↑14.80	20.53 ↑6.26
SkyQwen	33.48	18.44	0.00 ↓31.99	67.00 ↑47.20	3.33 -	3.33 ↑3.33	15.70 ↑9.42	21.25 ↑6.98
SkyCritic	39.07	26.95	51.71 ↑19.72	58.00 ↑38.20	**20.00 ↑16.67**	6.67 ↑6.67	13.00 ↑6.72	12.32 ↓1.95
GPT-5-Judge	45.85	35.75	91.96 ↑59.97	56.20 ↑36.40	16.67 ↑13.34	10.00 ↑10.00	21.52 ↑15.24	18.17 ↑3.90
Human	44.40	34.42	**95.30 ↑63.31**	53.80 ↑34.00	**20.00 ↑16.67**	10.00 ↑10.00	8.52 ↑2.24	18.89 ↑4.62
RLAR(Ours)	**47.16**	**38.24**	93.70 ↑61.71	59.80 ↑40.00	**20.00 ↑16.67**	**13.33 ↑13.33**	**23.31 ↑17.03**	19.30 ↑5.03

Table 4: **Main Results (Translation, General & Stats)**. **Bold**/underline: best/second-best. ↑/↓: changes vs. Zero-Shot baselines. **LLM-J**: LLM-judge. **UChat**: ULTRACHAT. The **bold** dataset (**WMT24**) is unseen during training. **Avg.**: overall average across all four task domains (shared with Table 3). **T&G**: local average over Translation & General benchmarks in this table only.

Domain	**Avg.**	**T&G**	**Translation**				**General**	**Stats**
Dataset	**Score**	**Avg.**	Flore		**WMT24**		UChat	Avg Len
Metric	-	-	BLEU-2	LLM-J	BLEU-2	LLM-J	LLM-J	#
Llama-3.1-8B								
Base	37.81	54.58	21.04	77.27	26.39	79.24	68.97	152.84
SkyLlama	23.47	40.88	21.16 ↑0.12	45.32 ↓31.95	23.23 ↓3.16	54.68 ↓24.56	60.03 ↓8.94	147.39
SeedX	17.29	24.95	0.00 ↓21.04	23.24 ↓54.03	0.00 ↓26.39	42.76 ↓36.48	58.73 ↓10.24	89.95
SkyQwen	31.68	55.52	26.42 ↑5.38	77.53 ↑0.26	29.80 ↑3.41	78.74 ↓0.50	65.13 ↓3.84	134.21
SkyCritic	34.05	45.96	25.31 ↑4.27	42.53 ↓34.74	28.21 ↑1.82	67.82 ↓11.42	65.93 ↓3.04	106.97
GPT-5-Judge	39.89	**56.99**	20.35 ↓0.69	**82.50 ↑5.23**	26.37 ↓0.02	**84.42 ↑5.18**	**71.32 ↑2.35**	203.31
Human	33.56	53.37	26.61 ↑5.57	70.67 ↓6.60	**29.81 ↑3.42**	75.88 ↓3.36	63.87 ↓5.10	76.76
RLAR(Ours)	**41.73**	55.59	**26.73 ↑5.69**	74.83 ↓2.44	29.80 ↑3.41	79.54 ↑0.30	67.03 ↓1.94	141.85
Qwen3-8B								
Base	29.12	60.22	28.46	84.00	32.50	84.46	71.70	1064.04
SkyLlama	29.25	61.13	28.78 ↑0.32	85.59 ↑1.59	33.75 ↑1.25	86.54 ↑2.08	71.00 ↓0.70	844.96
SeedX	35.14	61.31	30.34 ↑1.88	84.97 ↑0.97	35.25 ↑2.75	85.14 ↑0.68	70.83 ↓0.87	1135.82
SkyQwen	33.48	61.58	29.70 ↑1.24	85.72 ↑1.72	33.82 ↑1.32	85.18 ↑0.72	73.47 ↑1.77	1192.12
SkyCritic	39.07	61.37	28.66 ↑0.20	86.09 ↑2.09	32.52 ↑0.02	86.46 ↑2.00	73.10 ↑1.40	691.97
GPT-5-Judge	45.85	**63.59**	28.31 ↓0.15	**89.77 ↑5.77**	32.64 ↑0.14	**88.93 ↑4.47**	**78.32 ↑6.62**	1462.29
Human	44.40	62.32	30.71 ↑2.25	85.11 ↑1.11	35.91 ↑3.41	85.22 ↑0.76	74.63 ↑2.93	692.71
RLAR(Ours)	**47.16**	63.03	**31.12 ↑2.66**	86.49 ↑2.49	**36.07 ↑3.57**	86.67 ↑2.21	74.81 ↑3.11	986.23

average generation lengths across all benchmarks to detect potential reward-hacking behaviors associated with the verbosity bias inherent in LLM-based rewarding. GPT-5 is utilized as the primary evaluator for all LLM-as-a-judge assessments, and the 0-10 scoring range is normalized to 0-100 in accordance with other metrics. We provide an **average score** across all benchmarks, where the translation score is a composite of 0.5 BLEU-2 and 0.5 LLM-judge.

4.2 EXPERIMENTAL SETUP

Base Models: We use Qwen3-8B (Yang et al., 2025) and Llama-3.1-8B-Instruct (Weerawardhena et al., 2025). To examine various reward system architectures, we incorporate the following baselines:

Classification Reward Baselines: For the single classification reward model setting, we select `Skywork-Reward-V2-Llama-3.1-8B` (**SkyLlama**) and `Skywork-Reward-V2-Qwen3-8B` (**SkyQwen**) (Liu et al., 2024), both of which represent the state-of-the-art on REWARDBENCH-V1/V2 (Lambert et al., 2024; Malik et al., 2025b). Additionally, we incorporate `Seed-X-8B-RM` (**SeedX**) due to its robust performance in multilingual contexts (Cheng et al., 2025).

Generative Reward Baselines: We evaluate `Skywork/Skywork-Critic-Llama-3.1-8B` (**SkyCritic**) (Tu et al., 2024), which generates natural language justifications with its judgments. Furthermore, we employ GPT-5-as-a-judge (**GPT-5-Judge**) as a high-capacity competitor, following the RL from AI Feedback paradigm (Lee et al., 2024). In this setup, GPT-5 is prompted to act as an evaluator, and its numerical scores are extracted as reward signals. The specific prompt template is detailed in Appendix C.2.

Human Baseline: We implement a task-specific reward system to calibrate performance against optimal human-defined heuristics. For each task, the reward function is defined by the metrics most appropriate for the domain. Specifically, we utilize **NEM** for mathematics:

$$R_{\text{math}}(r, \hat{a}) = \begin{cases} 1, & \text{if NEM}(r, \hat{a}) \\ 0, & \text{otherwise} \end{cases}$$

where $\hat{a}$ denotes the ground-truth numerical answer, and **pass@1** for coding:

$$R_{\text{code}}(r) = \begin{cases} 1, & \text{if Exec}(r) = \text{Pass} \\ 0, & \text{otherwise} \end{cases}$$

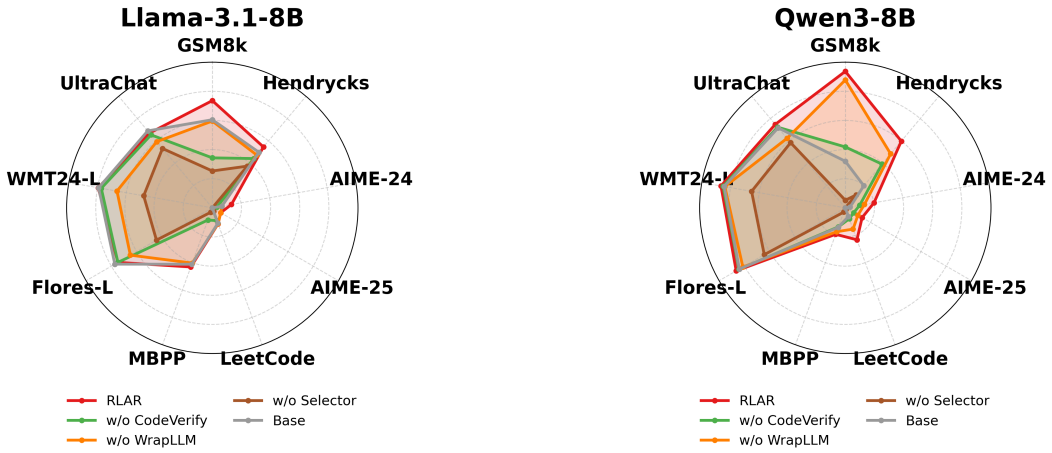
For translation, the reward is calculated as a hybrid score $w_1 \cdot \text{BLEU} + w_2 \cdot \text{SeedX}$, where the scaling factor is $w_1 = 1/2, w_2 = 1/2$ and SeedX's score is normalized to $(0, 1)$. The dialogue tasks are rewarded only by **SkyLlama**. We provide the **training configurations** in Appendix C.1 and **hyperparameters** for training in Appendix C.3. RLAR is powered by GPT-5 by default.

4.3 MAIN RESULTS

Superiority of RLAR across Mixed Domains. RLAR consistently outperforms both classification-based and generative reward baselines across multiple tasks and backbones (average score $+10.4\%$ on Llama and $+61.9\%$ on Qwen). Notably, RLAR achieves the best results on most benchmarks (73.6 NEM on GSM8k for Llama-3.1 and 23.3 Pass@1 on LeetCode for Qwen3) without significant degradation in other distributions. Improvements are also observed on the OOD benchmarks: AIME-24($+16.6$), AIME-25($+13.3$), MBPP($+5.03$), and WMT-24($+3.41$ BLEU). In contrast, **SkyLlama**, **SkyQwen**, and **SeedX** fail catastrophically with Qwen3 on GSM-8K (-31.9) and ULTRACHAT while benefiting HENDRYCKS and LEETCODE. This indicates that RLAR maintains stable and generalizable performance by dynamically modeling domain objectives into new reward designs without neglecting specific distributions.

On Llama-3.1, RL post-training shows limited headroom due to near-saturated base performance (60.5 on GSM8k, 41.17 on MBPP). Nevertheless, RLAR yields the largest gains on most benchmarks ($+13.1$ on GSM-8k, $+3.41$ on WMT BLEU). **SkyLlama**, **SkyQwen**, and **SkyCritic** largely fail on GSM8k ($-49.2, -44.3, -15.9$) and rarely show benefits on OOD benchmarks. **GPT-5-Judge** improved Llama on GSM8K by 11.4 but degraded HENDRYCKS-MATH by 6.60 and LEETCODE by 4.04, demonstrating RLAR's more robust reward design.

Comparison with GPT-5-Judge and Human Baseline. While **GPT-5-Judge** is competitive, it exhibits vulnerabilities in rule-heavy tasks: on Llama-3.1, it triggers drops on Hendrycks-Math and LeetCode (-6.60/-4.04) while RLAR does not (+4.80/+0.45), suggesting reward systems lacking explicit rule verification are susceptible to "blind spots." Furthermore, an 80-step (256 samples per step) RL session with GPT-5 costs ~120M API tokens and 288 GPU hours, whereas RLAR requires only 23.7M tokens and 72 GPU hours. Intriguingly, RLAR surpasses the Human baseline in several reasoning tasks (Math and Code on Llama, HENDRYCKS-MATH, LEETCODE on Qwen). Analysis reveals that RLAR's reward signals provide more comprehensive coverage than simple heuristics, incorporating additional test cases and formatting checks.



(a) Llama-3.1-8B (b) Qwen3-8B

Figure 3: Ablation studies on Llama-3.1 and Qwen3.

4.4 Reward Hacking Identification

Hack with Format. A critical failure mode observed in classification-based RM baselines on Qwen3-8B is "Format Hacking." Specifically, all three classification baselines (**SkyLlama, SeedX, SkyQwen**) collapsed to a 0.00 score on GSM8K. Manual inspection reveals that the policy models learned to bypass task-specific constraints: while the instructions required "####" before GSM8K answers and "`\boxed{}`" for HENDRYCKS-MATH, the models mistakenly unified all outputs into the Hendrycks format. Thus, baselines may overlook subtle instructional requirements, allowing the policy to "exploit" the reward system by mimicking specific output formats rather than optimizing the actual reward. Our method exhibits robustness against such loopholes, as the trained policy can correctly handle the formats across all benchmarks.

Hack with Verbosity. Furthermore, we observe a persistent verbosity bias in LLM-based reward baselines. For example, models trained with GPT-5-Judge consistently produce outputs 30% **longer** than those partly learned from rule-based rewards (Human/RLAR). This confirms that verbosity bias is inherently latent in reward LLMs and can be triggered during RL. Due to its rewarding versatility and the abundant included verifiable rewards, RLAR does not introduce a significant long-tail generation problem.

4.5 Ablations on Modules

To investigate the contribution of each component within the RLAR framework, we conduct an ablation study with the following configurations: (1) **w/o CodeVerify** and (2) **w/o WrapLLM**, which respectively disable the corresponding actions in the tool synthesis module; and (3) **w/o Selector**, where instead of immediate selection post-synthesis, a reward tool is randomly sampled only after all reward tools have been generated. Results of these variants are shown in Figure 3.

Removing either component of the tool synthesis module consistently results in degradation. Specifically, **without CodeVerify**-designed tools, mathematical reasoning and coding tasks (GSM8K, LEETCODE) drop significantly. **WrapLLM** is essential for chat and translation tasks. The above results confirm the necessity of query-specific reward designs, which is the core motivation of RLAR.

Moreover, the Random Reward Selection variant (**w/o Selector**) underscores the importance of reward matching. The policy degrades on almost all benchmarks. This confirms that appropriate reward selection is crucial in the overall RL training procedure, and that our framework effectively synthesizes high-quality reward tools superior to random alternatives.

We further analyzed the WrapLLM search engine design. The average retrieved model position is 5.64 on the first page, indicating high relevance. **Error analyses are in Appendix** F.1.

5 Related Work

5.1 LLM RL Reward Designs

Discriminative reward models provide human-preference oracles for RL alignment (Ouyang et al., 2022; DeepSeek-AI et al., 2025; Liu et al., 2024). **Generative** rewards extend classification to generation in mathematics (Zhang et al., 2025a) and RLHF (Ke et al., 2024; Wang et al., 2024; Zhu et al., 2025; Li et al., 2023), and can be integrated with reasoning methods such as CritiqueGRPO (Zhang et al., 2025b). **Verifiable** rewards are efficient for mathematics and code, where explicit rules support binary correctness checks (Shao et al., 2024; Lambert et al., 2025). Chang et al. (2025) extend this approach with standard NLP metrics for instruction-following alignment.

5.2 RL from AI Feedback

RLAIF reduces reliance on manual reward annotations (Lee et al., 2024); self-rewarding uses the policy to evaluate its own generations (Yuan et al., 2025); and LLM-as-a-judge uses a strong model with an evaluation prompt (Zheng et al., 2023). RewardAgent combines pre-specified reward designs through an LLM (Peng et al., 2025). These approaches retain human priors in their prompts or reward specifications. RLAR extends reward-design flexibility through tool synthesis and evaluates explicit rewards within GRPO, rather than DPO.

5.3 Dynamic Reward Assigning

Existing approaches refine rewards from policy traces through weight pruning and reward shaping (Afonso et al., 2025; Carta et al., 2022), curriculum scheduling, or prompt-guided reward schedules (Xie et al., 2025; Singla et al., 2024). RLAR instead searches online models and generates code to **design new reward functions**. Its per-query routing supports **cross-domain optimization**, where sub-domains require substantially different reward designs.

6 Conclusion

In this paper, we present RLAR, a novel agentic rewarding framework that designs query-specific reward functions for RL algorithms. By integrating a self-evolving tool library with dynamic synthesis and selection, RLAR achieves superior performance across diverse domains, including math, code, translation, and general chat. Beyond empirical gains, RLAR demonstrates strong robustness against format and verbosity reward hacking. By significantly reducing the costs compared to RLAIF practices, RLAR establishes the "LLM as Reward Designer" paradigm.

ETHICS STATEMENT

Our research aims to provide positive societal impacts by improving the reliability of reward system design for the reinforcement learning framework in LLM post-training. However, we acknowledge potential negative societal impacts. Like many tuning methods, our approach could theoretically be repurposed for malicious uses, such as exposing malicious reward model checkpoints to RLAR and thereby planting intended deficiencies in the policy LLM. Furthermore, if RLAR is deployed without careful calibration, it might inherit or amplify biases present in the training data, leading to unfair outcomes for marginalized tasks. We encourage future research to focus on bias mitigation and robustness evaluation before deploying such methods in production environments.

We have carefully considered the dual-use nature of our work. Because our released code poses a potential risk of misusing public model checkpoint repositories such as HuggingFace and ModelScope, we have implemented the following safeguards.

Originality and Attribution: The core originality of this work lies in the design and implementation of the RLAR framework. The training outcomes and performance improvements presented in this paper are the combined result of the RLAR architecture and the foundational capabilities of the pre-trained checkpoints we utilized. We explicitly attribute these capabilities to the original authors of the checkpoints, all of which have been appropriately cited. To further ensure transparency, we have documented our execution logs and provided them in Appendix E.

Safeguards and Infrastructure Protection: To prevent potential misuse of our codebase and protect public infrastructure, we have included explicit usage guidelines in our repository and licensing terms. Crucially, we have disabled automated checkpoint downloading features by default. This design choice prevents our codebase from being exploited to generate excessive or unnecessary network requests (unintentional DDoS) against external model hosting platforms. Furthermore, the external repositories hosting these checkpoints require authenticated user login to utilize their client APIs. This reliance on authentication provides an additional, robust layer of security to prevent the abuse of public network and storage resources.

REPRODUCIBILITY STATEMENT

The task mixture and dataset sizes are specified in Section 2.1 and Table 1, with dataset descriptions and filtering details in Appendix B. Section 4.2 and Appendix C.1 describe the experimental setup; Appendix C.3 lists training hyperparameters. Prompts for the GPT-5-based components and evaluator are provided in Appendices D and C.2.

AI USE STATEMENT

During data preparation and evaluation, we used GPT-5 to filter low-quality samples, generate reasoning outputs and reference answers where needed, and serve as an automated judge for a subset of benchmark scores, including translation and general dialogue evaluations. These AI-based scores are reported separately from rule-based metrics such as numeric exact match, unit-test pass rates, and BLEU.

Within RLAR, GPT-5 supports reward-tool retrieval, synthesis, and selection, including the generation of code-based verifiers and model-deployment wrappers, as described in the methodology. During manuscript preparation, generative AI assisted with translation, language polishing, draft feedback, typesetting, and formatting checks.

We have reviewed the AI-assisted work described above and take responsibility for the final content of the paper and accompanying materials.

REFERENCES

António Afonso, Iolanda Leite, Alessandro Sestini, Florian Fuchs, Konrad Tollmar, and Linus Gisslén. Self-correcting reward shaping via language models for reinforcement learning agents in games, 2025. URL https://arxiv.org/abs/2506.23626.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

Zheng Cai, Maosong Cao, Haojiong Chen, Kai Chen, Keyu Chen, Xin Chen, Xun Chen, Zehui Chen, Zhi Chen, Pei Chu, Xiaoyi Dong, Haodong Duan, Qi Fan, Zhaoye Fei, Yang Gao, Jiaye Ge, Chenya Gu, Yuzhe Gu, Tao Gui, Aijia Guo, Qipeng Guo, Conghui He, Yingfan Hu, Ting Huang, Tao Jiang, Penglong Jiao, Zhenjiang Jin, Zhikai Lei, Jiaxing Li, Jingwen Li, Linyang Li, Shuaibin Li, Wei Li, Yining Li, Hongwei Liu, Jiangning Liu, Jiawei Hong, Kaiwen Liu, Kuikun Liu, Xiaoran Liu, Chengqi Lv, Haijun Lv, Kai Lv, Li Ma, Runyuan Ma, Zerun Ma, Wenchang Ning, Linke Ouyang, Jiantao Qiu, Yuan Qu, Fukai Shang, Yunfan Shao, Demin Song, Zifan Song, Zhihao Sui, Peng Sun, Yu Sun, Huanze Tang, Bin Wang, Guoteng Wang, Jiaqi Wang, Jiayu Wang, Rui Wang, Yudong Wang, Ziyi Wang, Xingjian Wei, Qizhen Weng, Fan Wu, Yingtong Xiong, Chao Xu, Ruiliang Xu, Hang Yan, Yirong Yan, Xiaogui Yang, Haochen Ye, Huaiyuan Ying, Jia Yu, Jing Yu, Yuhang Zang, Chuyu Zhang, Li Zhang, Pan Zhang, Peng Zhang, Ruijie Zhang, Shuo Zhang, Songyang Zhang, Wenjian Zhang, Wenwei Zhang, Xingcheng Zhang, Xinyue Zhang, Hui Zhao, Qian Zhao, Xiaomeng Zhao, Fengzhe Zhou, Zaida Zhou, Jingming Zhuo, Yicheng Zou, Xipeng Qiu, Yu Qiao, and Dahua Lin. Internlm2 technical report, 2024.

Thomas Carta, Pierre-Yves Oudeyer, Olivier Sigaud, and Sylvain Lamprier. Eager: Asking and answering questions for automatic reward shaping in language-guided rl, 2022. URL https://arxiv.org/abs/2206.09674.

Yapei Chang, Yekyung Kim, Michael Krumdick, Amir Zadeh, Chuan Li, Chris Tanner, and Mohit Iyyer. Bleuberi: Bleu is a surprisingly effective reward for instruction following, 2025. URL https://arxiv.org/abs/2505.11080.

Shanbo Cheng, Yu Bao, Qian Cao, Luyang Huang, Liyan Kang, Zhicheng Liu, Yu Lu, Wenhao Zhu, Jingwen Chen, Zhichao Huang, Tao Li, Yifu Li, Huiying Lin, Sitong Liu, Ningxin Peng, Shuaijie She, Lu Xu, Nuo Xu, Sen Yang, Runsheng Yu, Yiming Yu, Liehao Zou, Hang Li, Lu Lu, Yuxuan Wang, and Yonghui Wu. Seed-x: Building strong multilingual translation llm with 7b parameters, 2025. URL https://arxiv.org/abs/2507.13618.

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

Nicholas Kluge Corrêa. Aira, 2023. URL https://github.com/Nkluge-correa/Aira.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan,

Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing chat language models by scaling high-quality instructional conversations. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3029–3051, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.183. URL https://aclanthology.org/2023.emnlp-main.183/.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset, 2021. URL https://arxiv.org/abs/2103.03874.

Pei Ke, Bosi Wen, Andrew Feng, Xiao Liu, Xuanyu Lei, Jiale Cheng, Shengyuan Wang, Aohan Zeng, Yuxiao Dong, Hongning Wang, Jie Tang, and Minlie Huang. CritiqueLLM: Towards an informative critique generation model for evaluation of large language model generation. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13034–13054, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.704. URL https://aclanthology.org/2024.acl-long.704/.

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

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training, 2025. URL https://arxiv.org/abs/2411.15124.

Harrison Lee, Samrat Phatale, Hassan Mansoor, Kellie Ren Lu, Thomas Mesnard, Johan Ferret, Colton Bishop, Ethan Hall, Victor Carbune, and Abhinav Rastogi. RLAIF: Scaling reinforcement learning from human feedback with AI feedback, 2024. URL https://openreview.net/forum?id=AAxIs3D2ZZ.

Junlong Li, Shichao Sun, Weizhe Yuan, Run-Ze Fan, Hai Zhao, and Pengfei Liu. Generative judge for evaluating alignment, 2023. URL https://arxiv.org/abs/2310.05470.

Chris Yuhao Liu, Liang Zeng, Jiacai Liu, Rui Yan, Jujie He, Chaojie Wang, Shuicheng Yan, Yang Liu, and Yahui Zhou. Skywork-reward: Bag of tricks for reward modeling in llms, 2024. URL https://arxiv.org/abs/2410.18451.

Saumya Malik, Valentina Pyatkin, Sander Land, Jacob Morrison, Noah A. Smith, Hannaneh Hajishirzi, and Nathan Lambert. Rewardbench 2: Advancing reward model evaluation, 2025a. URL https://arxiv.org/abs/2506.01937.

Saumya Malik, Valentina Pyatkin, Sander Land, Jacob Morrison, Noah A. Smith, Hannaneh Hajishirzi, and Nathan Lambert. Rewardbench 2: Advancing reward model evaluation, 2025b. URL https://arxiv.org/abs/2506.01937.

NLLB Team, Marta R. Costa-jussà, James Cross, Onur Çelebi, Maha Elbayad, Kenneth Heafield, Kevin Heffernan, Elahe Kalbassi, Janice Lam, Daniel Licht, Jean Maillard, Anna Sun, Skyler Wang,

Guillaume Wenzek, Al Youngblood, Bapi Akula, Loic Barrault, Gabriel Mejia Gonzalez, Prangthip Hansanti, John Hoffman, Semarley Jarrett, Kaushik Ram Sadagopan, Dirk Rowe, Shannon Spruit, Chau Tran, Pierre Andrews, Necip Fazil Ayan, Shruti Bhosale, Sergey Edunov, Angela Fan, Cynthia Gao, Vedanuj Goswami, Francisco Guzmán, Philipp Koehn, Alexandre Mourachko, Christophe Ropers, Safiyyah Saleem, Holger Schwenk, and Jeff Wang. No language left behind: Scaling human-centered machine translation. *arXiv preprint arXiv:2207.04672*, 2022.

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

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In Pierre Isabelle, Eugene Charniak, and Dekang Lin (eds.), *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pp. 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073135. URL https://aclanthology.org/P02-1040/.

Hao Peng, Yunjia Qi, Xiaozhi Wang, Zijun Yao, Bin Xu, Lei Hou, and Juanzi Li. Agentic reward modeling: Integrating human preferences with verifiable correctness signals for reliable reward systems. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15934–15949, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.775. URL https://aclanthology.org/2025.acl-long.775/.

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

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

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

Somanshu Singla, Zhen Wang, Tianyang Liu, Abdullah Ashfaq, Zhiting Hu, and Eric P. Xing. Dynamic rewarding with prompt optimization enables tuning-free self-alignment of language models. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 21889–21909, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.1220. URL https://aclanthology.org/2024.emnlp-main.1220/.

Shiwen Tu, Liang Zhao, Liu Chris Yuhao, Zeng Liang, and Liu Yang. Skywork critic model series. https://huggingface.co/Skywork, September 2024. URL https://huggingface.co/Skywork.

Yidong Wang, Zhuohao Yu, Wenjin Yao, Zhengran Zeng, Linyi Yang, Cunxiang Wang, Hao Chen, Chaoya Jiang, Rui Xie, Jindong Wang, Xing Xie, Wei Ye, Shikun Zhang, and Yue Zhang. PandaLM: An automatic evaluation benchmark for LLM instruction tuning optimization. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=5Nn2BLV7SB.

Sajana Weerawardhena, Paul Kassianik, Blaine Nelson, Baturay Saglam, Anu Vellore, Aman Priyanshu, Supriti Vijay, Massimo Aufiero, Arthur Goldblatt, Fraser Burch, Ed Li, Jianliang He, Dhruv Kedia, Kojin Oshiba, Zhouran Yang, Yaron Singer, and Amin Karbasi. Llama-3.1-foundationai-securityllm-8b-instruct technical report, 2025. URL https://arxiv.org/abs/2508.01059.

Yunhui Xia, Wei Shen, Yan Wang, Jason Klein Liu, Huifeng Sun, Siyue Wu, Jian Hu, and Xiaolong Xu. Leetcodedataset: A temporal dataset for robust evaluation and efficient training of code llms, 2025. URL https://arxiv.org/abs/2504.14655.

Tian Xie, Zitian Gao, Qingnan Ren, Haoming Luo, Yuqian Hong, Bryan Dai, Joey Zhou, Kai Qiu, Zhirong Wu, and Chong Luo. Logic-rl: Unleashing llm reasoning with rule-based reinforcement learning, 2025. URL https://arxiv.org/abs/2502.14768.

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.

Rui Yang, Xiaoman Pan, Feng Luo, Shuang Qiu, Han Zhong, Dong Yu, and Jianshu Chen. Rewards-in-context: Multi-objective alignment of foundation models with dynamic preference adjustment. *International Conference on Machine Learning*, 2024.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. Self-rewarding language models, 2025. URL https://arxiv.org/abs/2401.10020.

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction, 2025a. URL https://arxiv.org/abs/2408.15240.

Xiaoying Zhang, Hao Sun, Yipeng Zhang, Kaituo Feng, Chaochao Lu, Chao Yang, and Helen Meng. Critique-grpo: Advancing llm reasoning with natural language and numerical feedback, 2025b. URL https://arxiv.org/abs/2506.03106.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. URL https://openreview.net/forum?id=uccHPGDlao.

Lianghui Zhu, Xinggang Wang, and Xinlong Wang. JudgeLM: Fine-tuned large language models are scalable judges. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=xsELpEPn4A.

A LIMITATIONS

We primarily validate RLAR on heterogeneous tasks in text forms. Due to budget constraints, we did not extend the scope to multi-modal or audio tasks such as text-to-image generation. We believe this is a promising direction for future work. There is still room for further analysis on the scalability of the RLAR framework.

In practice, some repository READMEs may become outdated when reporting (such as claiming to be state-of-the-art at the time of writing). Though not directly caused by the design, RLAR is potentially vulnerable to README hacking, as our assumption is that most of these repository READMEs are trustworthy. We leave the development of more robust retrieval modules for future work.

Lastly, we focus on language models that are modeled as text classifiers. This is quite similar to practices in the industry, mainly aiming to save the computational cost of reward calculation. For generative reward models, our framework can support development on this basis; however, given the constraints of our experimental setup, we consider this to be outside the scope of the present work.

B DATA PROCESS DETAILS

B.1 DETAILED INTRODUCTION OF DATASETS

Our online RL experiments cover mathematics, code, translation, and general dialogue. Table 1 specifies the training and evaluation sizes. The datasets and their roles are as follows.

Mathematical Reasoning: We use GSM8K (Cobbe et al., 2021) and HENDRYCKS-MATH (Hendrycks et al., 2021) for training and evaluation. Both require reasoning toward a final numerical answer, evaluated by numeric exact match. AIME-24/25 supplies additional evaluation-only problems for testing generalization beyond the training sources.

Code Generation: We use LEETCODE-DATASET (Xia et al., 2025) for training and evaluation, asking the model to complete cloze-style function implementations. Correctness is measured by unit-test execution. MBPP (Austin et al., 2021) is an evaluation-only benchmark for Python program synthesis, reported using pass@1.

Translation: We sample FLORES-200 (NLLB Team et al., 2022) across 20 directed translation pairs among Chinese, German, English, French, and Japanese. WMT-24 is reserved for evaluating generalization to unseen translation contexts. Translation quality is evaluated with BLEU and GPT-5-as-a-judge scores; the aggregate translation score combines BLEU-2 and the LLM-judge score with equal weights.

General Dialogue and Instruction Following: We use ULTRACHAT (Ding et al., 2023) for training and evaluating multi-turn dialogue and instruction following. GPT-5 serves as the evaluator, using the prompt in Appendix C.2 and scores normalized to the 0–100 scale.

Data Preparation: GPT-5 filters training samples with low-quality queries or responses using the prompt below. We then downsample the training sources to form the mixture in Table 1. GPT-5 also produces reasoning outputs and supplies reference answers when human-annotated reasoning references are unavailable. The evaluation-only datasets do not contribute training examples.

Reward Selection Validation: Separately from online RL training, REWARDBENCH-V2 (Malik et al., 2025b) is used to assess reward-tool orchestration and selection, as described in Section 3.5.

B.2 DATA FILTERING PROMPT

DATA_FILTER_PROMPT

Input: `task_samples`, `quality_standards`

You are given a set of task samples, each consisting of:

1. **User Query** – the task or request made to the model.
2. **Model Response** – the output given by the model.

The samples may come from mathematical reasoning, code generation, translation, or general dialogue and instruction-following tasks.
Your Goal: Identify and select only the samples that did not meet quality standards based on:

A. Query Quality Issues:

- Ill-formed or incomplete queries; Ambiguous or misleading instructions.
- Irrelevant or off-topic requests; Grammatically broken or nonsensical input.

B. Response Quality Issues:

- Incorrect or factually wrong answers; Incomplete responses.
- Poor language quality or incoherent writing.
- Hallucinations or made-up facts; Misinterpretation of the query.

Instructions:

1. For each sample, examine both the query and response.
2. Mark the sample as **"Fail"** if either the query quality or the response quality is below standard.
3. Briefly explain why the sample fails, citing issues in query, response, or both.
4. Output only the failing samples, in the format:

```
[Sample ID]
Query:  ...
Response:  ...
Fail Reason:  ...
```

Be strict in applying the criteria — even if only one side (query or response) is substandard, the sample should be considered as failing.

C Experiment Training Details

C.1 Training Configuration

For the SFT setting, we tuned the base model on the training dataset for 2 epochs. All RL experiments were conducted using the GRPO (Shao et al., 2024) algorithm framework for 80 steps in total. Since Qwen3-8B is a reasoning model, we enforced a formatting constraint: the reward was set to zero if the reasoning chain failed to be properly encapsulated within '<think>' and '</think>' tags. We did not apply a length penalty during RL training, as generation length does not consistently correlate with task performance. More training details are provided in Appendix C.3.

All experiments were performed on a cluster of 8×NVIDIA H100 GPUs (80GB), using a global batch size of 256 and mixed-precision (FP16) training. An auxiliary cluster of 8×NVIDIA A100 GPUs (80GB) was dedicated to the deployment and inference of the reward models.

C.2 Prompt for the LLM Judge

LLM-as-a-judge Prompt

Input: `prompt`, `candidate`, `reference`

You are an expert evaluator of language model outputs. You will receive:

1. **Prompt:** The original instruction/task given to the model.
2. **Candidate Response:** The model's output to be evaluated.
3. **Reference Response:** A high-quality gold-standard or reference output.

Your task:

- Evaluate the quality of the *Candidate Response* compared to the *Reference Response* and in relation to the given *Prompt*.
- Consider the task category (**translation**, **summarization**, **generation**, **infilling/cloze**, **conditional generation**, **math**, or **instruction following**) and adjust criteria accordingly.
- Score on a scale from **0 to 10** according to the rubric below.
- Output the score in the format `[[X]]` (where X is the integer) **once**, followed by a clear explanation.

EVALUATION DIMENSIONS BY TASK CATEGORY

(Use whichever are relevant to the given prompt)

- **Translation**: Accuracy, fidelity, fluency, grammar, style.
- **Summarization**: Coverage, factual faithfulness, conciseness, coherence.
- **Generation**: Relevance, originality, creativity, coherence, style.
- **Infilling/Cloze**: Correctness of missing content, contextual fit, fluency.
- **Math/Reasoning**: Correctness of logic, clarity, rigor of explanation.
- **Instruction Following**: Alignment with intent, completeness.

SCORING RUBRIC (0–10)

10: Perfect. Fully correct, faithful, and clear. No errors.

8–9: Very Good. Minor style issues or tiny, overlookable omissions.

6–7: Good/Fair. Mostly correct but with notable small issues or some loss of clarity.

4–5: Poor/Borderline. Mix of correct/incorrect; noticeable gaps; low reliability.

1–3: Very Poor. Incoherent, irrelevant, or almost entirely wrong.

0: No meaningful output or completely unrelated.

OUTPUT FORMAT

Respond with:

```
[[X]]
Explanation:[Detailed explanation within 50 words, citing
criteria.]
```

[Prompt]
`{prompt}`

[Candidate Response]
`{candidate}`

[Reference Response]
`{reference}`

C.3 Hyperparameters Details for RL Training

We use the volcano engine reinforcement learning for LLMs framework, VERL (Sheng et al., 2024). We validate the implementation of the framework and run all our RL experiments based on it. Below are the hyperparameters for all our experiments and we use the same set of hyperparameters for all experiments.

Table 5: Hyperparameter settings for GRPO training with Qwen3-8B.

Category / Parameter	**Value**
Training & Data	
Total Epochs	5
Global Train Batch Size	256
Max Prompt Length	10,000
Max Response Length	5,000
Truncation Strategy	'error'
Filter Overlong Prompts	True
Algorithm (GRPO/PPO)	
Advantage Estimator	GRPO
Learning Rate	1×10^{-6}
PPO Mini-batch Size	16
KL Coefficient	0.0001
KL Loss Type	low_var_kl
Entropy Coefficient	0
KL in Reward	False
Rollout (vLLM Engine)	
Rollout Samples (n)	8
Max Batched Tokens	65,536
GPU Memory Utilization	0.7
Prompt/Response Length	10k / 5k
Parallelism & Infrastructure	
Nodes / GPUs per Node	1 / 8
Tensor Parallel (TP) Size	2
Pipeline Parallel (PP) Size	4
Gradient Checkpointing	True
Micro Batch Size (Actor)	1

The other hyperparameters, such as optimizer β, are set to the defaults from the framework trainer configurations at https://github.com/volcengine/verl/tree/main/verl/trainer/config.

D Prompt Details

D.1 Prompt for Task Decomposition

Task_Decomposition_Prompt

Input: original_task

Please break down the following generative task into a combination of several basic generative tasks:
Basic task list: 1. Controlled generation: Generate coherent natural language text that meets certain given conditions. Best for simple, clear tasks; complex writing should be split into smaller steps like planning and cloze generation.
2. Translation: Generate a corresponding text in another natural language from a text in one natural language.
3. Text summarization: Summarize the given text, retaining the main information.

Table 6: Hyperparameter settings for GRPO training with Llama-3.1-8B.

Category / Parameter	**Value**
Training & Data	
Total Epochs	5
Global Train Batch Size	256
Max Prompt Length	10,000
Max Response Length	5,000
Truncation Strategy	'error'
Filter Overlong Prompts	True
Algorithm (GRPO)	
Advantage Estimator	GRPO
Learning Rate	1×10^{-6}
PPO Mini-batch Size	16
KL Coefficient	0.0001
KL Loss Type	low_var_kl
Entropy Coefficient	0
KL in Reward	False
Rollout (vLLM Engine)	
Rollout Samples (n)	8
Max Batched Tokens	65,536
GPU Memory Utilization	0.7
Prompt/Response Length	10k / 5k
Parallelism & Infrastructure	
Nodes / GPUs per Node	1 / 8
Tensor Parallel (TP) Size	2
Pipeline Parallel (PP) Size	4
Gradient Checkpointing	True
Micro Batch Size (Actor)	1

4. Question answering: Provide appropriate answers based on background information and question requests provided by the user.
5. Paraphrasing: Modify the provided text into a different form of expression that meets the given rewriting requirements.
6. Cloze generation: Given a continuous piece of text with missing parts, generate appropriate text for the missing positions so that the original text becomes complete, coherent, and consistent.
7. Planning generation: Plan a high-level outline in order to accomplish a relatively complex generative task, such as creating a chapter list, designing character traits, designing scripts, or designing a timeline.
8. Code: Generate executable code that meets the specified requirements, or supplement or revise code according to the given requirements. The defining criterion for this task is that the output is primarily code.
Decomposition goal:
- Break down the complex generative task provided by the user into a list composed of the above basic tasks according to its logical steps. - Steps should be arranged in execution order, and the description should start from the original input form and proceed until the task is completed.
- Each step must clearly specify the "basic task type" and the execution content of that step.
- If the task does not need to be broken down, provide a single-step basic task and rewrite its description into a clearer instruction that aligns with the type of task in the basic task list.
Output format requirements:
- List the decomposition results step-by-step (step number + basic task type name + specific execution description).
- Enclose the final result within <Result> ... < \Result > tags.
Below is an example:

[Example Start]
Task to be decomposed: Please provide an English summary for the following Chinese document.
Decomposition result:
1. Translation: Please translate the following Chinese document into an English document.
2. Text summarization: Please summarize the given English document, and ensure the summary does not exceed 200 words.
[Example End]
Now, perform the above decomposition process on the given question (or task description) below, and write the final decomposition result within <Result> ... < \Result > tags.
{original_task}

D.2 Prompt Details for Reward Model Choice

D.2.1 Search API Tool

```
{
   "type": "function",
   "function": {
      "name": "search_serper_engine",
      "description": "Performs a Google search using the Serper API
          restricted to finding Hugging Face model checkpoints. Use
          this tool only to look up Hugging Face checkpoint URLs,
          model pages, or related information. Short queries work best.
           Reward model might be confusing with base models or chat
          models",
      "parameters": {
         "type": "object",
         "properties": {
            "query": {
               "type": "string",
               "description": "The search query for Hugging Face
                   checkpoints, e.g., model names or keywords to
                   locate on huggingface.co."
            }
         },
         "required": ["query"]
      }
   }
}
```

D.2.2 Prompt for search results filtration

Search Results Filtration

Input: `original_task`

You are given a list of search engine results with position IDs.
Your task is to filter them according to the following rules:

1. **Identify Reward Models:**
   - Keep only results that are **reward model** links.
   - Reward models often have model names containing keywords like `-Reward-` or `-RM-`.
   - Discard results for base models (`-Base`), instruct models (`-Instruct`), or chat models (`-Chat`).
   - If a model name has none of these hints, and it's unclear whether it is a reward model, discard it.

2. **Hugging Face Model Repositories Only:**
   - Keep only links pointing to **Hugging Face model repositories**.
   - Discard datasets, research papers, blog posts, or other non-model content.
3. **Score Output Format only:**
   - Regression models only, in other words, models that output a score (e.g., 0–1) rather than generating text.

Directly discard those items that violate rule 1, 2, or 3 and keep the rest. Output the remaining items in a list using their original position IDs like `[0, 1, 3, 5, ...]`. If none of the items are left, output an empty list `[]`.

{results}

D.2.3 Prompt for search results Reranking

Search Results Reranking

Input: `original_task`

You are given a list of search engine results with position IDs. Your task is to filter and rank them according to the following rules:

1. **Identify Reward Models:**
   - Keep only results that are **reward model** links.
   - Reward models often have model names containing keywords like `-Reward-` or `-RM-`.
   - Discard results for base models (`-Base`), instruct models (`-Instruct`), or chat models (`-Chat`).
   - If a model name has none of these hints, and it's unclear whether it is a reward model, discard it.
2. **Hugging Face Model Repositories Only:**
   - Keep only links pointing to **Hugging Face model repositories**.
   - Discard datasets, research papers, blog posts, or other non-model content.
3. **Score Output Format only:**
   - Regression models only; in other words, models that output a score (e.g., 0–1) rather than generating text.

Directly discard those items that violate rule 1, 2, or 3 and keep the rest. Output the remaining items in a list using their original position IDs, sorted by relevance, like `[0, 1, 3, 5, ...]`. If none of the items are left, output an empty list `[]`.

{results}

D.2.4 Prompt for search results LLM-based Reward Model Implementation

Reward Tool Implementation

Input: original_task

Implement a python script for launching a reward model according to the following informative scripts. The model local checkpoint is {model_local_dir}. The cuda device for the model is "{cuda_device}". You should write a function, that support input parameter: - prompt: str,

instruction or context conditions - response: str, the text need to be evaluated - reference: str, some reference answer/response for the above prompt
Your implementation are free to use the packages mentioned in the scripts. Name the calculation function starting with "compute_", such as "def compute_XXX(...)" where XXX should be the reward model name or related abbreviation. Make sure the model checkpoint is loaded precisely once in the script. Format your output enclosed within "python \ n xxxx \n". Also, additionally print the calculation funciton after four sharp marks ####, such as "#### def compute_xxx(...)" in the end of your output (outside the python script).
{scripts}
[your implementation]

D.3 Prompt for Code-Agent workflow

D.3.1 Plan

LIST_TASK_PROMPT

Input: `{task}`

You are an expert in designing reward models and evaluation metrics for the **{task}** task. Your goal is to list **3–5 possible reward model or evaluation metric choices** for this task, drawing from the following two categories:

1. **Rule-based** – Explicit rules (e.g., exact match with reference output, length constraints) used directly as rewards.
2. **Metric-based** – Standard NLP metrics (e.g., BLEU, ROUGE, METEOR) used to evaluate and reward generated results.

Output formatting requirements:

- Place your results **after four hash marks (####)**.
- For **each choice**, indicate its **category** and **name**, using the format:

  `#### <Category>/<Name>: <Brief description>`
- Use a **new line** for each choice.

Example:

```
#### Metric-based/BLEU: Measures the n-gram overlap between
generated output and reference text.
#### Rule-based/Length: Rewards outputs within the target
length range for conciseness.
```

D.3.2 Write

`WRITE_CODE_PROPMT`

Implement the following metric according to description using python. Write a function begin with `'compute_xxx'`. The function accepts: `prompt`, `candidate_response`, `reference_response`. Return a scalar score.
Output the python code in "'`python ...` "'. List requirements in `requirements.txt` style.

`[metric description]`

D.4 PROMPTS FOR WRAPLLM AND CODEVERIFY

TASK_CLS_PROMPT

You will be given a `question` and an `answer` from a language model interaction. Your job is to determine the main type of task being performed. Examples include but are not limited to: translation, summarization, question answering, code generation, math solving, creative writing, RLHF alignment, explanation, classification, etc. You do not need to list all possible task types — instead, use your judgment to give the most fitting label for this specific instance. **Rules:**

1. Output only the task type as a concise label (maximum **three words**).
2. Do not include any extra text, punctuation, or explanation.
3. If uncertain, choose the closest-fitting description.

Provide your output results after four sharp marks `####`, such as `"#### Translation"`.
Input example:

```
"question": "You are a helpful assistant
that translates English sentences to French...
[English Input]\n Michael Gill formed The Murder
Mile... [French Output]\n",
"answer": "Michael Gill a formé The Murder
Mile..."
```

Output: `#### English-French Translation`
Now classify the given instance according to these rules.

TASK_DECOMP_PROMPT

Please break down the following generative task into a combination of several basic generative tasks:
Basic task list:

1. **Controlled generation:** Generate coherent natural language text that meets certain conditions.
2. **Translation:** Generate corresponding text in another natural language.
3. **Text summarization:** Summarize the given text.
4. **Question answering:** Provide appropriate answers based on background info.
5. **Paraphrasing:** Modify text into a different form of expression.
6. **Cloze generation:** Fill in missing parts of a text.
7. **Planning generation:** Plan a high-level outline (chapter lists, scripts).
8. **Code:** Generate or revise executable code.

Decomposition goal:

- Break down the complex task into a list of the above basic tasks.
- Steps should be in execution order.
- Each step must specify the "basic task type" and execution content.

Output format requirements:

- List results step-by-step (step number + type + description).
- Enclose the final result within `<Result> ... </Result>` tags.

`{original_task}`

LIST_TASK_PROMPT

You are an expert in designing reward models and evaluation metrics for the **{task}** task. Your goal is to list **3–5 possible reward model or evaluation metric choices**, drawing from:

1. **Rule-based** – Explicit rules (e.g., exact match, length constraints).
2. **Metric-based** – Standard NLP metrics (e.g., BLEU, ROUGE).

Output formatting requirements:

- Place your results **after four hash marks (####)**.
- Format: `#### <Category>/<Name>: <Brief description>`
- Use a **new line** for each choice.

WRITE_CODE_PROPMT

Implement the following metric according to description using python. Write a function begin with `'compute_xxx'`. The function accepts: `prompt`, `candidate_response`, `reference_response`. Return a scalar score.
Output the python code in "'`python ... "'. List requirements in `requirements.txt` style.

`[metric description]`

WRAP_TOOL_PROMPT

Write a short description of the following reward function. Briefly explain what it calculates and how to use it for NLP evaluation.
NAME: `{name}`
CODE:
"'`python
{code}
"'
Write your description after `####`, e.g., `"#### This function calculates..."`.

METRIC_DESIGN_PROMPT

You are an expert in designing evaluation metrics with python. Implement `{metric}`.
Info: `{info}`
Write a function `def compute_XXX(prompt, candidate_response, reference_response)` that returns the score. Enclose in "'`python ... "'.

REWARD_MODEL_DESIGN_PROMPT

Expert in reward models. Write a script to calculate reward scores. Function: `def compute_XXX(prompt, response, reference)`. Device: `"cuda:0"`. Load model precisely once.
README.md: `{readme}`
Path: `{model_path}`

RERANK_PROMPT

Filter search results based on:

1. **Identify Reward Models:** Keep `-Reward-` or `-RM-`. Discard `-Base`, `-Instruct`, `-Chat`.
2. **Hugging Face Only:** Model repositories only. Discard datasets/papers.

3. **Score Output:** Regression models (e.g., 0–1).

Output original IDs as a list: `[0, 1, 3]`.

`{results}`

`LLMRM_IMPLEMENT_CODE`

Implement python script for reward model.
Checkpoint: `{model_local_dir}`
Device: `"{cuda_device}"`
Function: `def compute_XXX(prompt, response, reference).`
Format: ```` ```python ... ``` ````. Print function signature after `####` at the end.

`{scripts}`

`SELECT_INFORMATIVE_FILES`

Identify informative files from HF repo:
`{file_list}`
Output list: `["file1.py", "README.md"]`. No extra text.

E GENERATED TOOLS

Table 7: A list of the generated reward function tool names by our code-agent.

Type	Metric	Type	Metric
rule_based	Forbidden_Words	rule_based	Stepwise_Completeness
rule_based	Prompt_Adherence	rule_based	Length
rule_based	Numeric_Accuracy	rule_based	Exact_Template_Match
rule_based	Novelty_Penalty	rule_based	Contradiction_Detection
rule_based	Disallowed_Phrase_Penalty	rule_based	Exact_Output_Match
rule_based	No_Unsupported_Claims	rule_based	Reference_Match
rule_based	Exact_Answer_Match	rule_based	Named_Entity_Preservation
rule_based	Unit_Consistency	rule_based	Keyword_Presence
rule_based	Minimal_Edit_Distance	rule_based	Thesis_Inclusion
rule_based	Mandatory_Content_Inclusion	rule_based	Scientific_Claims_Match
rule_based	Pronounceability	rule_based	Position_Sensitivity
rule_based	Section_Coverage	rule_based	Entity_Presence
rule_based	Answer_Type_Match	rule_based	Stepwise_Correctness
rule_based	Terminology_Accuracy	rule_based	Diversity_Score
rule_based	Forbidden_Content	rule_based	Fact_Match
rule_based	Forbidden_Phrase_Detection	rule_based	No_Information_Leakage
rule_based	Annotation_Completeness	rule_based	Grammar_and_Spelling_Accuracy
rule_based	Clarity_Constraint	rule_based	Answer_Presence
rule_based	No_Overlap_with_Input	rule_based	No_Syntax_Errors
rule_based	Numeric_Tolerance	rule_based	Edit_Distance
rule_based	Keyword_Coverage	rule_based	No_Repetition
rule_based	Length_Ratio	rule_based	One-Hot_Accuracy
rule_based	Novelty	rule_based	Exact_Match
rule_based	Pattern_Compliance	rule_based	Step_Match
rule_based	Syntax_Validity	rule_based	Format_Compliance
rule_based	Allowed_Vocabulary	rule_based	Entity_Overlap
rule_based	Explicit_Irrelevance	rule_based	Accuracy
rule_based	Coverage_of_Key_Points	rule_based	Section_Presence
rule_based	Clarity	rule_based	Test_Case_Pass_Rate
rule_based	Dictionary_Filtering	rule_based	Length_Expansion
rule_based	Content_Inclusion	rule_based	Error_Pattern_Removal
rule_based	Plagiarism_Check	rule_based	Functionality_Test
rule_based	Politeness_Constraint	rule_based	Formatting_Compliance
rule_based	Exact_Test_Case_Pass	rule_based	Key_Information_Coverage
rule_based	Genre-Adherence	rule_based	Passes_Unit_Tests
rule_based	Exact_Step_Match	rule_based	Exact_Keyword_Match
rule_based	Required_Field_Inclusion	rule_based	Attribute_Coverage

Type	Metric	Type	Metric
rule_based	Valid_Vocabulary	rule_based	Medical_Term_Coverage
rule_based	Keyword_Absence	rule_based	Required_Component_Presence
rule_based	Final_Answer_Correctness	rule_based	Keyword_Inclusion
rule_based	Structure_Compliance	rule_based	Step_Consistency
rule_based	Readability	rule_based	No-Answer_Accuracy
rule_based	Length_Constraint	rule_based	Error_Reduction
rule_based	Answer_Type_Mismatch	rule_based	Thesis_Presence
rule_based	Case-Insensitive_Match	rule_based	Topic_Divergence
rule_based	Exact_Numeric_Match	rule_based	Originality-Penalty
rule_based	Keyword_Exclusion	rule_based	Structure
rule_based	Format_Consistency	rule_based	Required_Elements
rule_based	Reference_Citation	rule_based	Instruction_Match
rule_based	Key_Concepts_Inclusion	rule_based	Stepwise_Solution_Match
rule_based	Fact_Consistency	rule_based	Step_Count_Constraint
nlp_metric	F1_Score	nlp_metric	METEOR
nlp_metric	ROUGE	nlp_metric	GLEU
nlp_metric	BERTScore	nlp_metric	M^2_Score
nlp_metric	chrF	nlp_metric	ROUGE-L
nlp_metric	Levenshtein_Distance	nlp_metric	BLEU
nlp_metric	Distinct-n	nlp_metric	CodeBLEU
model_based	Content_Novelty_Score	model_based	Negative_Relevance_Score
model_based	Topic_Classifier	model_based	Perplexity

Table 8: Successfully Top@9 deployed LLM-based reward models.

Source	Repo Name
(Liu et al., 2024)	Skywork/Skywork-Reward-V2-Llama-3.1-8B
(Liu et al., 2024)	Skywork/Skywork-Reward-V2-Qwen3-8B
(Liu et al., 2024)	Skywork/Skywork-Reward-V2-Llama-3.2-3B
(Liu et al., 2024)	Skywork/Skywork-Reward-V2-Qwen3-4B
(Cheng et al., 2025)	ByteDance-Seed/Seed-X-RM-7B
	OpenAssistant/reward-model-deberta-v3-base
(Yang et al., 2024)	Ray2333/gpt2-large-helpful-reward_model
(Corrêa, 2023)	nicholasKluge/RewardModel
(Cai et al., 2024)	internlm/internlm2-1_8b-reward
(Malik et al., 2025a)	allenai/Llama-3.1-8B-Base-RM-RB2

Table 9: Unmatched Task Types by Web Retrieval

Task Type	Unmatched(%)
Infilling	47.4
Essay Generation	43.8
Multi-Turn	8.8

F Additional Analysis

F.1 Error and Robustness Analysis

We conducted an error analysis by counting the task types of instructions for which the Web-Agent could not find a specialized reward model (unmatched conditions). The breakdown in Table 9 shows that the majority of unfound instructions originate from essay infilling/generation tasks. Specifically, there is currently no corresponding reward model explicitly trained for these two task domains, which accounts for the high unmatched ratio in these categories. Notably, when a specialized tool is unmatched, RLAR defaults to using a generic, default LLM-based reward model (`skywork-llama`).

To assess the robustness of the searching module, we tracked the average item position (calculated as page rank $\times 10$) for the matched reward model. Across all sampled categories, the overall average retrieval position was 5.64 items. As detailed in Table 10, all individual sub-categories consistently found the optimal item on the first page, confirming the robustness and high precision of the agent's query generation and search logic.

We further validate the soundness of the framework's design by including a detailed analysis of the generated tool quality (Appendix F.2) and an investigation into the reward tool usage within our main experiments (Appendix F.3). In summary, code-agents achieve a 94.9% executable rate when utilizing rule/metric-based tools, and we observe a dominant percentage of LLM-based reward tool usage in text-generation tasks.

F.2 Reward Tool Generation Quality

We evaluate the quality of reward tools produced by our two agents for tool generation mainly along their construction validity and summarized in Table 11.

Code-agent tools. Across all the training queries, the code agent generated **118** reward scripts, among which **112** (**94.9%**) were directly executable under our standardized interface.[3] By type, the set comprises **102** rule-based functions (**86.4%**), **12** standard metric implementations (**10.2%**; e.g., BLEU, METEOR), and **4** learned-model–based scorers (**3.4%**). Rule-based tools typically encode

[3]Executability is checked by importing the generated function, calling it with a minimal synthetic triplet `(prompt, candidate, reference)` and verifying a numeric return type without exceptions.

Table 10: Average Retrieval Position Ranks

Category	Avg Pos
Summ	7.17
Translation	2.36
RLHF	5.03
Multi-Turn	7.61
Infill/Gen	3.75
Math	6.87

Table 11: Summary of reward tool generation outcomes.

Category	Count
Code-agent scripts (total)	118
Executable	112 (94.9%)
Rule-based	102 (86.4%)
Metric-based	12 (10.2%)
Learned-model–based	4 (3.4%)
Web-agent repos (retrieved)	21
Deployed	10 (47.6%)
Rejected (size)	2
Rejected (not classification)	6
Rejected (insufficient docs)	3

task-specific verifiable criteria (e.g., numeric-consistency checks for GSM8K or explicit-irrelevance penalties for RLHF-style preference items), while metric-based tools provide length- or n-gram–aware surrogates for general text quality. We discard learned-model–based proposals from the code agent since they are potential out-of-memory threats to the deploying server.

Web-agent tools. The web agent retrieved **21** candidate repositories from public model hubs (primarily Hugging Face and ModelScope) that matched the predicted task label and satisfied our *reward-model* filter. The filter eliminates base/instruct/chat/vision models and retains text-classification-modeled reward models with download access. After automatic screening and wrapping, **10** repositories (**47.6%**) were successfully deployed behind a uniform Python API. The remaining **11** were rejected due to: model size prohibitive for our inference node (**2**), non–text-classification architectures (**6**), or insufficient/ambiguous repository documentation for reliable wrapping (**3**).

The high executability of code-agent tools (**94.9%**) and the moderate but reliable deployment rate of web-agent tools (**47.6%**) indicate that RLAR can *consistently* materialize task-aligned reward functions across heterogeneous inputs.

F.3 Reward Tool Usage and Selection Patterns

Having established that RLAR can reliably generate and deploy reward tools, we next examine *how* these tools are actually invoked during training. This analysis addresses two questions: (i) which categories of tools dominate in practice, and (ii) how the usage patterns vary with task source and affect the learned policy.

We plot the actual usage of tools by examining the tool matching conditions based on the data source in the training set, as shown in Figure 4. Across all $8{,}000+$ training samples, the majority of calls are routed to **LLM-based reward models** (**96.4%**), while **rule-based** and **metric-based** tools are invoked only sparsely. The most frequently selected individual model is `Skywork/Skywork-Reward-V2-Llama-3.1-8B`, accounting for **52.5%** of calls. A significant proportion of samples fall back to rule-based numeric-consistency checks ("explicit number match"). On translation tasks, the web-agent-originated `Seed-X-RM-7B` dominates, capturing cross-lingual adequacy more effectively than generic reward models.

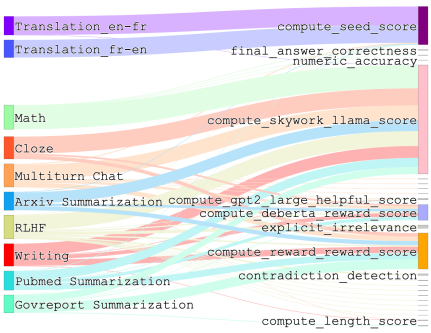


Figure 4: Matching tools with source training dataset distribution.

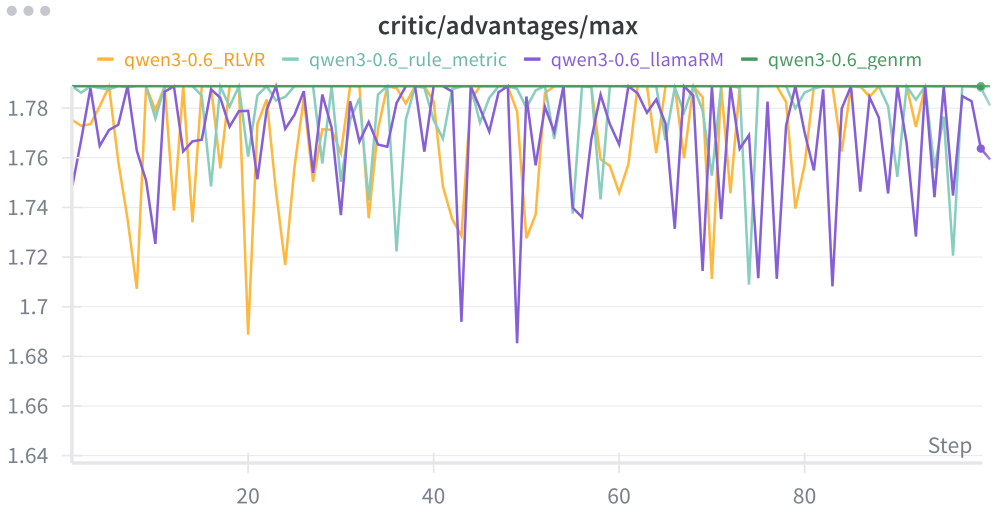


Figure 5: Maximum Advantage Estimations

The dominance of LLM-based rewards suggests that, for heterogeneous open-domain training, high-capacity discriminative models remain the most trusted. Nevertheless, the occasional use of rule-based checks in math and RLHF tasks demonstrates that RLAR is capable of *combining expert heuristics* when appropriate. RLAR does not rely on a single global reward model but instead orchestrates a *portfolio* of evaluators aligned with each domain. As shown in the previous subsection, this diversity translates into smoother advantage estimation and stronger updates during policy optimization.

F.4 Impact on Advantage Estimation and Policy Learning

We examine the records from the Qwen experiments covering Generative RM, RLAR, and single generic reward, regarding the estimated min/max of advantage per step (Figure 5 and Figure 6), and calculate the proportion that triggered clipping. Higher rates of being clipped mean a higher absolute value of estimated advantage. From the results, for Generative RM, rollouts triggering both upper-clip and under-clip occur in every update step. Compared to single generic reward, RLAR has a significantly higher clipping rate. This is direct evidence that **methods with better performance tend to estimate larger advantages in absolute values**.

Returning to the discussion of Advantage Estimation $\hat{A}_i = \frac{r_i - \text{mean}(r)}{\text{std}(r)}$. Consider two types of reward functions in Figure 7: the blue one is sensitive to extreme values (smaller variance) while the orange one is evenly modeled (higher variance). Assuming uniform roll-out sampling, a higher value of $\hat{A}_i$

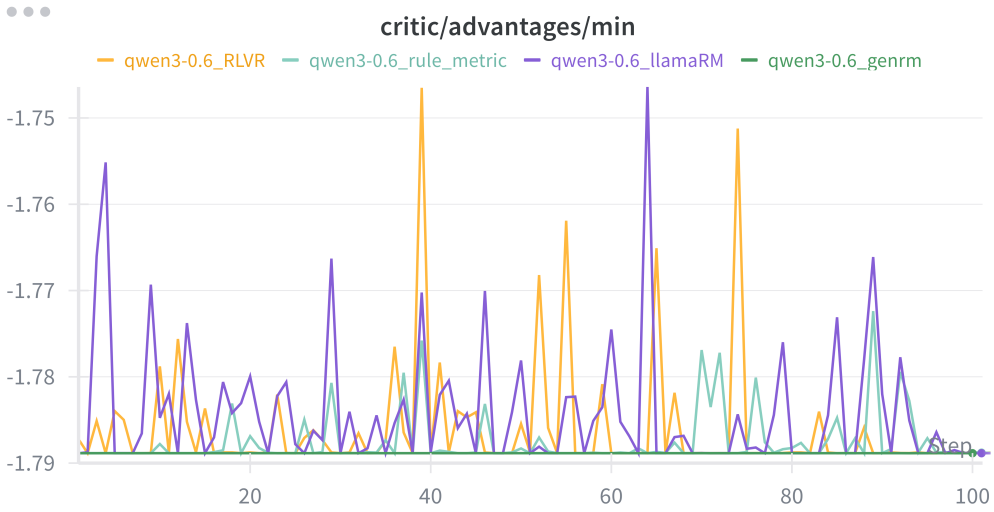


Figure 6: Minimum Advantage Estimations

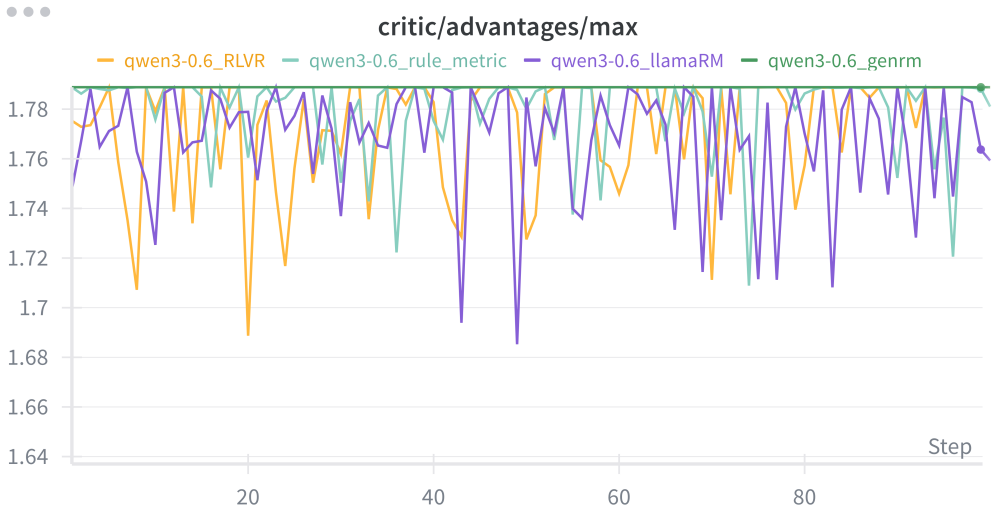


Figure 7: An illustration on the sensitivity to extreme values of reward functions.

suggests that the underlying reward function resembles **the sensitive type** (blue line). Therefore, extreme values (maximum/minimum) are divided by a smaller variance, resulting in more frequent reaching of the clip threshold. This is expected for policy optimization, where more weight should be assigned to these deviated rollouts as part of the exploration-exploitation balance.