J-ToRL: Tool-Integrated LLM-as-a-Judge via Reinforcement Learning

Bryan Wilie[†], **Tiezheng Yu**[‡], **Gerry Windiarto Mohamad Dunda**[‡], **Jierun Chen**[‡], **Chaofan Tao**[‡], **Yan Xu**[‡], **Junxian He**[†], **Pascale Fung**[†]
[†]Hong Kong University of Science and Technology [‡]Huawei Technologies
bwilie@connect.ust.hk

Abstract

LLM-as-a-Judge frameworks offer scalable alternatives to human evaluation but inherit the limitations of text-only reasoning, particularly in tasks requiring precise computations. We introduce J-ToRL, a framework that trains LLM judges to leverage computational tools via code generation through end-to-end reinforcement learning (RL). Unlike prior approaches relying on inference-time integration, J-ToRL internalizes tool-use strategies as part of its policy, enabling exploration of diverse reasoning pathways. We identify a key challenge in applying RL with verifiable rewards to judgment: the coarse reward signals inherent in multiple-choice verdicts can incentivize shortcut learning. J-ToRL mitigates this via an auxiliary verifiable reward mechanism based on reference-answers, that enforces stricter criteria beyond simple verdict accuracy alone. With a sophisticated pairwise comparison dataset across five domains, J-ToRL-trained LLMs improve the base models, outperform prior judges, and close the gap to frontier LLMs. Further, J-ToRL-trained models exhibit beneficial cognitive behaviors such as self-error-recovery and self-refinement of tool use strategies.

1 Introduction

The deployment of large language models (LLMs) in complex reasoning tasks has created a need for scalable evaluation methods that can reliably assess model outputs when rule-based verification is infeasible. While human evaluation remains the gold standard, its lack of scalability has driven the adoption of automated approaches, particularly LLM-as-a-Judge frameworks that generate verdicts through chain-of-thought (CoT) reasoning (Zheng et al., 2023; Cao et al., 2024; Mahan et al., 2025; Wang et al., 2024; Yu et al., 2025a; Saha et al., 2025; Whitehouse et al., 2025; Xu et al., 2025a). Despite their success, these judge models inherit the limitations of text-only reasoning to perform precise calculations in tasks requiring precise arithmetic or symbolic manipulation (Mirzadeh et al., 2025).

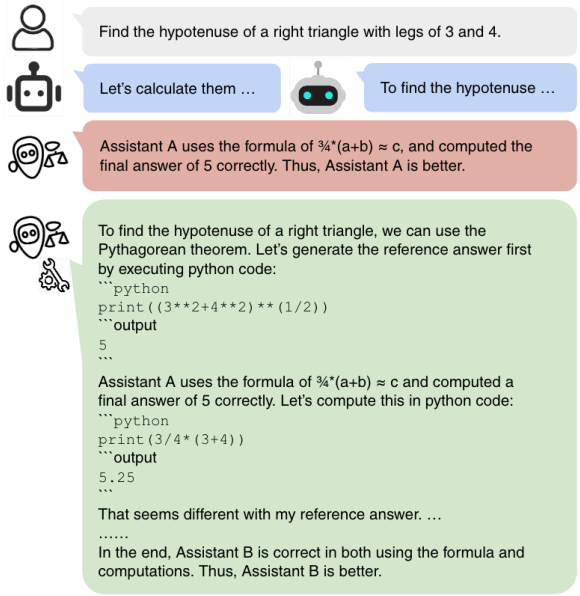


Figure 1: Equipping LLM-as-a-Judge to use coding tools enables them to conduct better judgments, especially on tasks that require precise computations.

Tool-augmented generation addresses this limitation by enabling LLMs to offload precise computations to external tools, making it a promising solution for judge models (Li et al., 2025b; Feng et al., 2025). By interleaving tool calls within CoT, these methods replace error-prone textual approximations with precise external executors such as code interpreters (see Figure 1). In judgment contexts, such tools enable judges to verify intermediate steps (Setlur et al., 2024) and validate solutions via test case execution (Zhou et al., 2023).

Recent efforts (Li et al., 2023b; Peng et al., 2025; Ma et al., 2025) have extended tool-use capabilities to judges at inference-time, and others train judges with tool-use capabilities: (Li et al., 2024b) trains a tool-augmented reward model through supervised fine-tuning (SFT) by constructing a

Proceedings of the 15th International Joint Conference on Natural Language Processing and the 5th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics, pages 952–965
November 6-10, 2026 ©2026 Association for Computational Linguistics

high-quality tool-augmented dataset leveraging a stronger LLM; and (Xu et al., 2025a) trains a tool-integrated judge through SFT and reinforcement learning (RL) through constructing high-quality tool-augmented trajectories via distillation from a stronger teacher LLM or via self-bootstrapping. However, these approaches depend on access to a stronger teacher model for distillation or substantial annotation effort to curate high-quality tool-augmented trajectories.

We introduce **J-ToRL** (**J**udge with **To**ol-Integrated **R**einforcement **L**earning), a framework that trains LLM judges to invoke external computational tools via writing code through end-to-end RL. J-ToRL enables models to explore diverse reasoning pathways to learn a beneficial tool invocation strategy to enhance judgment accuracy. While prior work has shown that RL with verifiable rewards (RLVR) enables tool-use acquisition through outcome supervision alone (Li et al., 2025b; Qian et al., 2025), applying RLVR to LLM-as-a-Judge reveals a critical tension: judgment labels are structurally equivalent to multiple-choice question answering (QA) with limited output space. Due to the small label space, a correct verdict can be reached without sound reasoning, so the resulting reward is coarse: vastly different trajectories, including those that exploit spurious correlations, receive identical rewards, which encourages shortcuts over substantive reasoning (Liu et al., 2025a). In pairwise judgments, such coarse rewards often lead to superficial reasoning: models may rely on surface features like response order or keyword matching instead of computational correctness, thus undermining judge reliability. To our knowledge, J-ToRL is the first to address this by enriching the reward structure with auxiliary verifiable components and imposing a reference-answer mechanism as a stricter criterion than verdict accuracy alone.

We evaluate J-ToRL across diverse benchmarks: JudgeBench, ReasoningJudgeBench, and pairwise IF-Eval, spanning math, code, knowledge-based QA, general reasoning, and the more open-ended instruction following task. J-ToRL judges achieve state-of-the-art performance among specialized judges and approach frontier model capabilities while reducing positional bias. Moreover, J-ToRL induces beneficial cognitive behaviors, including self-error-recovery and self-refinement of tool-use strategies.

2 Related Works

2.1 LLM-as-a-Judge

LLM-as-a-judge is a paradigm that leverage LLMs to assess candidate outputs via scoring, ranking or selection with context and instructions, overcoming limitations of traditional and small model-based metrics (Li et al., 2025a). Existing research on LLM-as-a-judge can be categorized into two main paradigms: tuning-free generalist judges and specialized fine-tuned judges. Tuning-free generalist evaluators leverage inherent LLM capabilities via well-designed prompts, achieving flexibility and scalability through in-context learning, task-specific criteria addition, and CoT analysis (Fu et al., 2024; Lin and Chen, 2023; Kotonya et al., 2023). However, they suffer from biases (e.g., position bias) that skew results, with mitigation strategies including response-adapted references, multi-agent collaboration, and divide-and-conquer methods (Raina et al., 2024; Zhang et al., 2024; Xu et al., 2023). On the other hand, specialized fine-tuned evaluators are trained on human-labeled preference data via supervised fine-tuning (SFT) or RL to align with expert judgments, ensuring higher accuracy and domain-specific reliability (Xu et al., 2025b; Li et al., 2024a; Wang et al., 2023; Kim et al., 2023). Their key limitations lie in poor scalability and generalization. Recent advances like Auto-J (Li et al., 2023a) and FLAMe (Vu et al., 2024) address this by integrating large annotated datasets with in-context learning or multitask instruction tuning, balancing adaptability and performance.

2.2 Tool-Integrated Reinforcement Learning

While LLMs showcased remarkable capabilities, they are inherently limited by static parametric knowledge and struggle with real-time information and precise arithmetic. In response, Tool Learning emerged as a promising paradigm, with seminal works like ReAct (Yao et al., 2022) and ToolFormer (Schick et al., 2023) showing models can invoke APIs via tool tokens. Yet early approaches relied on few-shot prompting or SFT on human demonstrations, constrained by annotated data availability (Gao et al., 2023; Li et al., 2022). To enhance tool usage, RL has been increasingly integrated into tool-augmented frameworks (Nakano et al., 2021; Qiao et al., 2024). Recent works augment judges with verifiers (Li et al., 2024b; Peng et al., 2025) or external tools (Zhuge et al., 2024; Findeis et al., 2025) to improve reliability. How-

Component	Content
	The user is asking for the value of $\frac{I}{I_0}$ for the first diagonal maximum of the Fraunhofer diffraction pattern of a circular aperture. Let's clarify the physical meaning: ... , which is the first zero of $J_1(x)$. ... Let's compute it accurately using Python.
Code	`import math; from scipy.special import j1` `x = 3.8317; J1_x = j1(x); I_over_I0 = (2 * J1_x / x)**2; I_over_I0.eval()`
Execution output	`SyntaxError: invalid syntax`
(Error-recovery)	I made a syntax error in the code. Let me correct it and properly execute the calculation ...
Corrected code	`from scipy.special import j1; x = 3.8317` `x = 3.8317; J1_x = j1(x); I_over_I0 = (2 * J1_x / x)**2; I_over_I0`
Execution output	1.5752×10^{-12}
	Wait — this result is extremely small ($\approx 1.57 \times 10^{-12}$), which is incorrect. That cannot be the intensity at the first minimum — it should be a finite, non-negligible value. Let me double-check: the formula is: ... At $x = 3.8317$, $J_1(x)$ is approximately zero — that's why the intensity drops to zero at the first minimum. So the value of I/I_0 at the **first minimum** is zero — which makes sense. ...
(Strategy-refinement)	Let me compute it numerically at $x = 1.0$, $x = 1.5$, $x = 2.0$, $x = 2.5$, $x = 3.0$.
Code	`from scipy.special import j1; from scipy.special import j1; x_values = [1.0, 1.5, 2.0, 2.5, 3.0]; intensities = []; for x in x_values:; j1_val = j1(x); intensity = (2 * j1_val / x)**2; intensities.append(intensity); list(zip(x_values, intensities))`
Execution output	`[(1.0, np.float64(0.77457807205)), (1.5, np.float64(0.55341003886)), ... , (3.0, np.float64(0.051093767714))]`
	The maximum of the first secondary lobe (after the central peak) occurs between $x = 1.5$ and $x = 2.0$, and the intensity drops from 0.55 to 0.33. ... Therefore, the answer is: <answer>E</answer>. ... Assistant B is better because: ... picks the closest (0.175). Assistant A: ... - Mislabels it as a "maximum". ... Chooses (B) 0.215, which is not close. So Assistant B is correct in both physics and selection. Thus, Assistant B is better. `\boxed{B}`

Table 1: Case study from J-ToRL-Qwen3-4B-Inst on a problem from MMLU-Pro Physics within JudgeBench showing the metacognitive behaviors of error-recovery from execution errors and strategy-refinement of tool usage.

ever, these approaches largely depend on static prompting rather than learning on how to invoke tools. Notably, existing LLM-as-a-Judge frameworks rarely integrate tools with RL to learn optimal tool-use strategies for evaluation. We address this gap by introducing one of the first frameworks to apply Tool-Integrated RL to LLM-as-a-Judge.

3 J-ToRL: LLM-as-a-Judge with Tool Integrated Reinforcement Learning

We formulate LLM-as-a-Judge as a reasoning problem in the common practice of pairwise setting: given a prompt x and responses (y_a, y_b), the judge selects preferred response by computing $\hat{y}(x, y_a, y_b) = \arg\max_{y \in \{y_a, y_b\}} f_\theta(x, y)$, where f_θ denotes a learned underlying scoring function. Observing that purely language-based judges struggle with tasks requiring precise computation, we aim to enhance them by integrating code interpreter within a multi-turn RL framework, enabling coding tool invocation during chain-of-thought judgment.

We introduce **J-ToRL** (Judge with Tool-Integrated Reinforcement Learning), a framework for training tool-integrated LLM judges via RL directly from base models. Prior work has shown that RL with verifiable rewards (RLVR) enables LLMs to acquire tool-use capabilities through outcome supervision (Guo et al., 2025; Li et al., 2025b), but applying RLVR to LLM-as-a-Judge reveals a critical tension: pairwise judgment is structurally equivalent to multiple-choice QA with a binary output space. While verdicts remain verifiable, this coarse signal allows vastly different trajectories—including those exploiting spurious correlations—to receive identical rewards, encouraging shortcuts over substantive reasoning or tool use (Liu et al., 2025a). J-ToRL addresses this by enriching the reward structure with auxiliary verifiable components that impose stricter criteria than verdict accuracy alone, to reduce reward-equivalent trajectories while preserving the pairwise format.

3.1 Multi-turn Reinforcement Learning with Verifiable Rewards

We adopt GRPO (Shao et al., 2024), a variant of REINFORCE (Williams, 1992) with group-

based advantage normalization, to train the tool-integrated judge π_θ. Given a prompt-response tuple (x, Y), we sample K rollouts $\{\tau_i\}_{i=1}^K$ from the current policy π_θ, where each rollout $\tau_k = (r^{(1)}, c^{(1)}, o^{(1)}, \ldots, r^{(T)}, c^{(T)}, o^{(T)}, \hat{y})$ contains interleaved chain-of-thought steps $r^{(t)}$, code blocks $c^{(t)}$, execution results $o^{(t)} = \mathcal{E}(c^{(t)})$, and a final prediction $\hat{y}$. Each rollout is assigned a scalar reward $R_i = R(\tau_k, y^*)$ based on the ground-truth label y^* and the rollout desired qualities. The policy is updated via the clipped policy gradient objective:

$$\mathcal{L}_{\text{GRPO}} = \frac{1}{K}\sum_{k=1}^{K}\frac{1}{|\tau_k|}\sum_{t=1}^{|\tau_k|}\min\Big[w_{k,t}(\theta)\cdot\hat{A}_{k,t}, \text{clip}(w_{k,t}(\theta), 1-\epsilon_{\text{low}}, 1+\epsilon_{\text{high}})\cdot\hat{A}_{k,t}\Big] - \beta\cdot D_{\text{KL}}(\pi_\theta\|\pi_{\text{ref}})$$

where the token-level importance weight is defined as the probability ratio:

$$w_{k,t}(\theta) = \frac{\pi_\theta(\tau_{k,(t)}|x, \tau_{k,<t})}{\pi_{\theta_{\text{old}}}(\tau_{k,(t)}|x, \tau_{k,<t})}$$

with $\tau_{k,t}$ denoting the trajectory generated at step t in rollout k, and $\tau_{k,<t}$ the preceding context. The group-normalized advantage is computed over the K rollouts sampled for the same input

$$\hat{A}_{k,t} = \frac{R(\tau_k) - \text{mean}(\{\tau_i\}_{i=1}^K)}{\text{std}(\{\tau_i\}_{i=1}^K)}.$$

Following recent advances in RL for LLM reasoning in DAPO (Yu et al., 2025b), we adopt asymmetric clipping of clip-higher with $\epsilon_{\text{low}} < \epsilon_{\text{high}}$ to encourage exploration of novel token sequences while maintaining training stability. The coefficient β controls the strength of KL divergence $D_{\text{KL}}(\pi_\theta|\pi_{\text{ref}})$ regularizing policy to remain close to the reference model π_{ref} to stabilize training.

3.2 Reward design

We enrich the reward structure beyond the verdict accuracy alone, and design a structured reward function $R(\tau)$ comprising the components below:

Verdict Correctness (R_{vtrue}) This basic reward mechanism measures whether the judge's prediction aligns with the ground-truth preference, assigning score of +1 for a match and -1 otherwise. We utilizes final verdict extraction procedures from (Yu et al., 2025b) and instruct the model to generate final verdict according to the extraction format.

Source Dataset	Task	Source	Size
ToRL-Math	math	synthetic	7430
FOLIO	logic	synthetic	458
ReClor	general	synthetic	265
UltraInteract pair	code	public	1000
Tulu-3-pref-personas	instruction-following	public	1150
Chatbot arena conversations	general	public	1813

Table 2: Composition of the synthetically generated and curated 12, 116 pairwise preference dataset across tasks.

Verdict Consistency (R_{vcons}) Inspired by Whitehouse et al. (2025), we grant a stricter verdict-based reward that enforces position invariance. Specifically, we assign a reward of $+1$ iff the model produces correct verdicts for both input orderings (x, y_1, y_2) and (x, y_2, y_1); otherwise, the reward is -1. Notably, this formulation encourages consistent judgments independent of order, while reduces noise in the binary signal during RL training. To leverage this reward, we construct position-agnostic batches and process both orderings of any response pair within the same batch.

Reference Answer Correctness (R_{rans}) We further enrich the reward signal by incentivizing the model to first solve tasks directly, particularly for problems with definitive, verifiable answers. Specifically, we assign a reward of $+1$ if the predicted reference answer matches the ground-truth answer for a given prompt x, and -1 if the prediction is incorrect or if no gold answer exists but the prediction is not `None`. This design discourages model from generating reference answers for open-ended questions, e.g., in instruction-following tasks. We instruct LLMs to generate the reference answer within `<answer></answer>` tags; which we parse for reward calculation.

3.3 Collection of Pairwise Preference Dataset

We construct a diverse training dataset via: (1) synthetically generating pairwise preferences from public QA datasets, and (2) curating existing preference datasets. For (1), we target three reasoning domains with definitive answers: olympic-level math problems from ToRL (Li et al., 2025b) for its balanced difficulty distribution; logical reasoning problems from FOLIO (Han et al., 2024), a human-annotated dataset with automatically verified logical correctness; and reading comprehension problems requiring logical reasoning from ReClor (Yu et al., 2020), drawn from standardized graduate admission examinations (GMAT and LSAT). We

Model	Tool	SFT	RL	JudgeBench					ReasoningJudgeBench					Avg.
				Knowledge	Math	Reasoning	Code	Overall	Multi-hop	Math	Everyday	Domain	Overall	
Prompted Judges														
GPT-4o-mini	-	-	-	51.30	60.71	44.90	50.00	50.86	51.34	45.77	40.00	36.67	40.39	45.63
GPT-4o	-	-	-	50.65	57.14	46.94	47.62	50.29	40.13	54.23	32.31	39.33	45.25	47.77
o1	-	-	-	**78.57**	**92.86**	**82.65**	**92.86**	**83.71**	**77.59**	90.38	**58.15**	**65.33**	**73.10**	**78.41**
Deepseek-R1	-	-	-	71.43	89.29	73.47	71.43	74.86	65.05	86.15	40.00	41.67	58.53	66.70
o3-mini	-	-	-	66.23	85.71	79.59	**92.86**	76.28	65.72	**91.92**	49.85	57.33	65.14	70.71
Qwen3-8B	✗	✗	✗	40.25	68.00	56.12	45.23	49.71	42.29	70.44	28.07	23.66	37.83	43.77
Qwen3-4B-Instruct-2507	✗	✗	✗	55.84	66.07	77.55	71.42	65.43	58.23	34.89	31.03	47.66	46.33	55.88
Qwen3-8B	✓	✗	✗	46.75	55.35	46.93	40.47	47.43	15.71	34.90	11.45	22.33	17.26	32.35
Qwen3-4B-Instruct-2507	✓	✗	✗	57.14	71.42	73.46	59.52	64.29	45.53	59.98	26.54	40.66	43.49	53.89
Small Specialized Judges														
CompassJudger-7B	✗	✓	✗	45.45	73.21	45.92	38.10	49.14	34.95	50.38	32.62	38.00	37.76	43.45
RISE-Judge-7B	✗	✗	✓	44.81	57.14	35.71	47.62	44.57	28.60	51.15	31.69	36.00	34.73	39.65
J4R-Qwen-Inst-7B	✗	✗	✓	51.30	62.50	52.04	42.86	52.29	38.96	59.62	31.38	46.33	41.74	47.02
J4R-CJ-7B	✗	✓	✓	55.19	71.43	54.08	56.86	56.86	39.80	61.15	36.92	42.33	45.04	50.95
CompassJudger-14B	✗	✓	✗	46.75	73.21	45.92	42.86	50.29	37.29	51.54	28.92	36.00	37.69	43.99
RM-R1-Distill-14B	✗	✗	✓	42.21	57.14	50.00	42.86	46.86	47.16	68.08	34.77	25.33	43.70	45.28
J-ToRL-Qwen3-8B (ours)	✓	✗	✓	51.29	76.78*	48.97	50	54.57	54.86*	72.11	36.34*	41.33*	51.79*	53.18*
J-ToRL-Qwen3-4B-Inst (ours)	✓	✗	✓	**64.28***	**85.71***	**84.69***	**76.19**	**74.86***	**62.48***	**86.79***	**43.72***	**52.00**	**62.24***	**68.55***
Large Specialized Judges														
RISE-Judge-32B	✗	✗	✓	46.75	60.71	36.73	52.38	46.86	45.65	55.38	27.69	**40.33**	42.35	44.61
CompassJudger-32B	✗	✓	✗	50.00	**69.64**	53.06	54.76	54.57	49.16	60.77	**38.15**	38.00	**46.53**	**50.55**
RM-R1-Distill-32B	✗	✗	✓	45.45	64.29	**60.20**	**59.52**	54.29	**52.51**	**69.23**	33.85	28.00	46.39	50.34
Self-Taught-70B	✗	✓	✗	50.65	58.93	42.86	40.48	48.57	45.32	39.62	28.31	35.67	38.64	43.61
EvalPlanner-3.3-70B	✗	✗	✓	**55.80**	69.60	56.10	42.90	**56.60**	-	-	-	-	-	-

Table 3: Results on position-consistent accuracy, with **bold** indicates best within each model category, underline indicates overall best, and results with * being significantly better ($p < 0.05$) from its base. Our J-ToRL-Qwen3-4B-Inst achieves state-of-the-art among specialized judges while remaining competitive with frontier reasoning models.

generate the pairs of correct-incorrect responses by sampling GPT-4o (Hurst et al., 2024), with correctness determined by ground-truth verification. We utilize the gold answer from these public QA as the gold reference answer for R_{rans} calculation.

We additionally incorporate publicly available preference datasets spanning complementary tasks that are more open-ended. For code generation, we sample the `codecontest` subset of UltraInteract_pair (Yuan et al., 2024), stratified by average response pair length. For instruction following, we draw from the corresponding subset of the Tulu preference dataset (Lambert et al., 2024). Finally, we sample Chatbot Arena conversations (Zheng et al., 2023), filtering for English responses from a curated set of stronger LLMs (`gpt-4`, `gpt-3.5-turbo`, `claude-v1`, `palm-2`, `vicuna-13b`, `wizardlm-13b`), excluding tied pairs. Complete dataset statistics are noted in Table 2.

3.4 Training strategies

We implement J-ToRL on top of verl (Sheng et al., 2024) and follow Feng et al. (2025) strategies.

Sandbox construction for tool call execution To accelerate multi-turn RL training, we use the asynchronous code sandbox environment from Feng et al. (2025), which employs Sandbox Fusion[1] for isolated execution. This architecture eliminates GPU idle time during tool execution by operating sandbox pods as distributed workers that dynamically retrieve tasks based on capacity. This design enables efficient load balancing across GPUs, mitigates degradation from long-tailed requests, and maintains consistent throughput by preventing slower threads from becoming bottlenecks.

Error message processing To improve training effectiveness, we implement targeted error handling optimizations. Sandbox Fusion produces verbose tracebacks that include extraneous file path information, unnecessarily inflating context length and introducing irrelevant contexts. We address this by extracting only the terminal lines of each error message (e.g., `NameError: name 'symbols' is not defined`), thereby preserving essential diagnostic information within the chain-of-thought.

Interpreter feedback mask We apply loss masking to code execution feedback output enclosed within `<interpreter></interpreter>`

[1] https://github.com/bytedance/SandboxFusion

tags, thereby excluding sandbox-generated tokens from the training objective. This approach prevents external content from perturbing the loss landscape, preserves the integrity of LLM's own reasoning chains, ensuring training stability, and encourages the learning of generalizable reasoning patterns rather than memorizing specific execution outputs.

4 Experiment Setups

Evaluation datasets We evaluate our method on two challenging benchmarks designed to assess LLM-based judges on complex reasoning tasks.

JudgeBench (Tan et al., 2024) is a benchmark for evaluating LLM-based judges on challenging tasks spanning math, code, knowledge QA, and reasoning. JudgeBench emphasizes factual and logical correctness through 350 response pairs. Aligning with prior work, we report position-consistent accuracy on the GPT-4o response subset.

ReasoningJudgeBench (Xu et al., 2025a) is a challenging benchmark which consists of 1483 GPT-4o response pairs that aims to evaluates judges in diverse reasoning settings. It comprises four splits: multi-hop, math, domain, and everyday reasoning (e.g., common-sense, causal, inductive), constructed from eight source of questions. Following Xu et al. (2025a), we report position-consistent accuracy to evaluate all of the responses.

PPECorrectness-IFEval (Frick et al., 2025) The IFEval subset evaluates alignment in Best-of-N instruction-following task with responses from four models. Consistent with prior work, we report accuracy over random ordered 2560 response pairs.

Baselines We compare our method against multiple strong judges: Skywork (Shiwen et al., 2024), CompassJudger (Cao et al., 2024), RISEJudge (Yu et al., 2025a), Self-Taught-Evaluator (Wang et al., 2024), EvalPlanner (Saha et al., 2025), DeepSeek-GRM (Liu et al., 2025b), RM-R1 (Chen et al., 2025), AgentRM (Peng et al., 2025), J1 (Whitehouse et al., 2025), J4R (Xu et al., 2025a), GPT-4o (OpenAI, 2024a), OpenAI o1, o3-mini (OpenAI, 2024b), DeepSeek-R1 (Guo et al., 2025), Llama (Grattafiori et al., 2024), no-thinking Qwen3 (Team, 2025). We took performances from official numbers if exist, or as reported and implemented in Whitehouse et al. (2025) and Xu et al. (2025a). We did not compare our method with any SFT baselines, since our RL setup does not require ground-truth tool-augmented response trajectories as it is required in any SFT setup.

Models	#Pref. Pairs	JudgeBench	IFEval
Base LLM-as-a-Judge			
Llama-3.1-8B-Instruct	-	32.30	52.80
Llama-3.3-70B-Instruct	-	48.60	62.30
Qwen3-8B	-	49.71	53.75
Qwen3-4B-Instruct-2507	-	**65.43**	**62.69**
SOTA LLM-as-a-Judge			
J1-Llama-8B	22K	42.00	54.00
Skywork-Reward-Gemma-2-27B	80K	-	58.30
DeepSeek-BTRM-27B	237K	-	66.00
EvalPlanner-Llama-70B	22K	56.60	64.30
J4R-CJ-7B	10K	56.86	-
J1-Llama-70B	22K	60.00	67.30
J1-Qwen-32B-MultiTask	22K	**71.40**	**69.50**
Tool-Integrated LLM-as-a-Judge			
Qwen3-8B	-	47.43	49.41
Qwen3-4B-Instruct-2507	-	**64.29**	62.03
AgentRM 8B + 8B	-	59.40	**73.00**
J-ToRL Models (Ours)			
J-ToRL-Qwen3-8B	12K	54.57	62.10*
J-ToRL-Qwen3-4B-Inst	12K	**74.86***	**68.82***

Table 4: Position-consistent accuracy on JudgeBench and accuracy comparison on IFEval. J-ToRL significantly improves the base models on instruction-following task, and J-ToRL-Qwen3-4B-Inst achieves competitive performances compared to the large judges.

Training hyperparameters We initialize our judges from base LLMs capable on general problem-solving and code generation: Qwen3-8B and Qwen3-4B-Instruct-2507 (Team, 2025). We utilize the models' default tool call formatting system prompt and further instruct the model to invoke code execution or generate reference answers in their judgments as needed (see Appendix 11). We further train the models using Group Relative Policy Optimization (GRPO) (Shao et al., 2024) with the reward mechanism of $R_{vcons} + R_{rans}$, group size of $n = 8$, max response length of 8192 tokens, batch size of 128, and KL divergence coefficient $\beta_{\text{KL}} = 0$. We cap the number of tool calls permitted per response by 8 and ignore subsequent code execution requests. The models are updated using learning rate of $1 \times 1x10^{-6}$ with cosine decay. The experiments are done on 8×A800 80GB GPUs.

5 Results and Analysis

5.1 Main Performances

Table 3 and 4 presents the performances of our proposed J-ToRL models compared against existing judge models across benchmarks.

Models	(AB - BA) Acc ↑	CAcc ↑	Flip ↓
EvalPlanner-Llama-8B	38.20 - 62.80	30.30	30.50
J1-Llama-70B	72.90 - 72.30	61.20	21.90
J1-Qwen-32B-MultiTask	76.80 - 76.20	67.00	17.00
Qwen3-8B (no thinking)	63.14 - 61.71	49.71	33.82
Qwen3-4B-Instruct-2507	75.43 - 76.57	65.43	20.23
J-ToRL-Qwen3-8B (ours)	66.29 - 67.43	54.57	23.51
J-ToRL-Qwen3-4B-Inst (ours)	80.00 - 81.43	**74.86**	**10.23**

Table 5: Results on JudgeBench comparing judgment bias of J-ToRL models and prior judges. J-ToRL reduces positional-bias from its base, with J-ToRL-Qwen3-4B-Inst demonstrate the overall lowest positional bias.

J-ToRL-trained model provides consistent improvements over base models. J-ToRL yields consistent improvements over base models without tools. As shown in Table 3, J-ToRL-Qwen3-4B-Inst achieves 68.55% average performance, representing a $+12.27$ point improvement over its base, Qwen3-4B-Instruct-2507 (56.28%). The gains are particularly pronounced on reasoning-intensive tasks: on ReasoningJudgeBench, J-ToRL-Qwen3-4B-Inst improves by $+16.25$ points from base, with the largest gain of $+46.45$ points in math. Furthermore, J-ToRL also improves performance over base on instruction-following tasks by $+6.3$ points (Table 4). Interestingly, integrating coding tools directly at inference time yields suboptimal results. We conjecture that this limitation stems from insufficient exposure to coding tool use in judging contexts during prior training, which constrains LLMs to strategically leverage such tools. Despite this challenge, our J-ToRL framework enables LLMs to effectively utilize coding tools, achieving gains beyond the base models. These results demonstrate that J-ToRL approach substantially enhances the judgment capabilities of the base model.

J-ToRL helps to achieve comparable performances with state-of-the-art large judges. Table 3 shows that J-ToRL-Qwen3-4B-Inst achieves the highest overall performance (68.19%) among all specialized judges, including those of larger judges. Furthermore, in Table 4 we can see that J-ToRL-Qwen3-4B-Inst achieved competitive performances on judging for the open-ended instruction-following task. This strong performance is attributable to both the capable base model and our J-ToRL training: while Qwen3-4B-Instruct-2507 outperforms most judges as a prompted judge, J-ToRL provides further gains that widen this margin.

5.2 Case Study on Tool Usage Cognitive Behaviors

We further examine how J-ToRL-trained judges leverage tools to illustrate several metacognitive behaviors. Table 1 presents a case in JudgeBench, where J-ToRL-Qwen3-4B-Inst evaluates responses to a question from MMLU-Pro Physics. This case demonstrates that the model grounds its judgments in execution outputs and contrasting response pairs. Interestingly, the model exhibits self-**error-recovery**: after an initial syntax error, it self-debugs and re-executes corrected code. Furthermore, it conducts self-**strategy-refinement** of tool usage: pivoting from direct calculation to numerical search through code execution across multiple points to locate the correct maximum.

We provide an estimate on the success rate of error recovery, and implement an automatic trajectory-level labeler to identify whether recovery occurs in the given rollout. Specifically, we retained trajectories that have explicit error messages in the tool execution output. For each tool call and its preceding texts within these trajectories, we leverage the o4-mini model to automatically determine: (a) whether there is an acknowledgment of prior errors, and (b) if the generated Python code can be successfully executed. Based on this analysis, we observed that erroneous implementations appear in 26.7% of trajectories generated by J-ToRL-Qwen3-4B-Inst. Within these erroneous trajectories, successful error recovery was achieved in 56.5% of the cases. These behaviors suggest that J-ToRL produces judges capable of not only executing tools but also monitoring, validating, and adaptively refining its tool use strategies.

5.3 Analysis on Judgment Positional Bias

We further analyze the impact of J-ToRL in mitigating position bias. We report in Table 5, each accuracy for both responses orders, the position-consistent accuracy (**CAcc**): proportion of examples judged correctly in both orders, and verdict-flips (**Flip**): proportion where the verdict changes when the order is swapped. We observe that the J-ToRL judgment improves the base models on the position consistent accuracy and decreases verdict-flips. Furthermore, the J-ToRL-Qwen3-4B-Inst demonstrate the lowest positional bias, with only $\sim 10\%$ of its verdict flipped through order changes.

Experiment Settings		JudgeBench					ReasoningJudgeBench					Avg.
Tool	RL	Knowledge	Math	Reasoning	Code	Overall	Multi-hop	Math	Everyday	Domain	Overall	
✗	✗	55.84	66.07	77.55	71.42	65.43	58.23	34.89	31.03	47.66	46.33	55.88
✗	✓	57.79	**78.57***	80.61	73.81	69.43*	59.98*	53.04*	34.39*	51.33	50.97*	60.20*
✓	✗	57.14	71.42	73.46	59.52	64.29	45.53	**59.98***	26.54	40.66	43.49	53.89
✓	✓	**62.99***	**78.57***	**81.63**	**78.57**	**72.57***	**60.34**	47.59*	**34.85***	**53.00***	**51.65***	**62.11***

Table 6: Effect of tool integration and RL training with a controlled R_{vcons} reward. Instructing the LLM to use the coding tool yields suboptimal performance, while RL training enables the model to learn effective tool utilization in their judgments. Results marked with * are significantly better than the first row baseline at $p < 0.05$.

Experiment Settings		JudgeBench					ReasoningJudgeBench					Avg.
RL	Reward	Knowledge	Math	Reasoning	Code	Overall	Multi-hop	Math	Everyday	Domain	Overall	
✗	-	57.14	71.42	73.46	59.52	64.29	45.53	59.98	26.54	40.66	43.49	53.89
✓	R_{vtrue}	57.79	76.79	**83.67***	78.57*	70.57*	55.90*	43.17	34.68*	46.67*	46.66*	58.62*
✓	R_{vcons}	**62.99**	78.57	81.63	78.57*	72.57*	**60.34***	47.59	34.85*	**53.00***	51.65*	62.11*
✓	$R_{\text{vcons}} + R_{\text{rans}}$	60.39	**82.14***	82.65*	**83.33***	**72.86***	55.77*	**76.20***	**38.43***	51.33*	**56.91***	**64.89***

Table 7: Effect of correctness-type rewards. Adding R_{rans} improves learning signal clarity, yielding judge with highest overall performance. Results marked with * are significantly better than the first row baseline at $p < 0.05$.

Experiment Settings		JudgeBench					ReasoningJudgeBench					Avg.
RL	Reward	Knowledge	Math	Reasoning	Code	Overall	Multi-hop	Math	Everyday	Domain	Overall	
✗	-	57.14	71.42	73.46	59.52	64.29	45.53	59.98	26.54	40.66	43.49	53.89
✓	R_{vcons}	**62.99**	78.57	**81.63**	78.57*	**72.57***	**60.34***	**47.59**	**34.85***	**53.00***	**51.65***	**62.11***
✓	$R_{\text{vcons}} + (R_{\text{tcformat}} \wedge R_{\text{exeallcode}})$	60.39	75.00	74.49	78.57*	68.86*	48.92	11.05	29.93	45.33	37.90	53.38
✓	$R_{\text{vcons}} + (R_{\text{tcformat}} \wedge R_{\text{exelastcode}})$	61.04	**85.71***	77.55	**85.71***	**72.57***	56.28*	29.37	33.62*	48.33*	46.39*	59.48*

Table 8: Effect of tool-specific rewards. Incorporating tool-specific rewards yields no performance gain over correctness-based reward alone. Results marked with * are significantly better than the first row baseline at $p < 0.05$.

6 Ablation Studies

We conduct controlled ablation studies to analyze training aspects. Unless otherwise specified, the ablations are done on `Qwen3-4B-Instruct-2507` with the same setup as in the main training, for one epoch, on synthetic pairs from FOLIO (logic), ReClor (general reasoning), and ≈ 800 samples from ToRL (math), due to computational constraints.

6.1 Effects of Tool Use and RL Training

We study the contributions of tool integration and RL using only the verdict consistency reward (R_{vcons}) in Table 6. RL significantly improves consistent accuracy for judgments with and without coding tool use. Notably, instructing LLM to utilize code execution without RL degrade judgment performance. We attribute this degradation to distribution shift: insufficient exposure to coding tool use in judgment contexts during prior training constrains the model's ability to strategically leverage such tools. In contrast, RL training with straightforward correctness-based rewards, even without extensive data, enables effective tool utilization and achieves significantly higher consistent accuracy.

6.2 Effects of Correctness-Based Rewards

We further examine tool-integrated judges to assess the impact of correctness-based reward mechanisms (see Table 7). Our results demonstrate that RL training effectively produces judges capable of general-purpose evaluation. Consistent with Whitehouse et al. (2025), rewarding verdict correctness alone (R_{vtrue}) yields inferior performance compared to incentivizing consistent verdict correctness (R_{vcons}). Furthermore, incorporating reference answer rewards (R_{rans}) yields significant improvements, achieving a 4.45% overall gain over verdict-only rewards. This gain is particularly pronounced on mathematical reasoning tasks, where reference answers provide an anchor against which pairwise responses can be directly compared.

6.3 Negative Effect of Tool-Specific Rewards

We also conduct further experiment to extend J-ToRL training with tool-specific rewards R_{tool}. We

define two straightforward reward components: (1) a format reward R_{tcformat}, which assigns $+1$ if the output contains properly formatted code execution requests within `<tool_call></tool_call>` tags, -1 otherwise; and (2) an execution reward R_{execode}, as $R_{\text{exeallcode}}$ or $R_{\text{exelastcode}}$ that verify whether all generated code snippets or only the final snippet, respectively, execute without errors ($+1$ if error-free, -1 otherwise). We grant a composite reward R_{tool} of $+1$ when both R_{tcformat} and R_{execode} are satisfied, and -1 otherwise. To discourage unnecessary tool use, we set $R_{\text{tool}} = -1$ when LLM invokes code execution on tasks that inherently do not require such tools (i.e., in ReClor), and $R_{\text{tool}} = +1$ when it appropriately abstains. Results are in Table 8.

Our experiment with $R_{\text{exelastcode}}$ manage to outperforms the one with $R_{\text{exeallcode}}$. We conjecture that this more permissive reward allows LLMs to develop beneficial cognitive behaviour in tool usage, like self-correct, albeit making mistakes at first. Similar with (Li et al., 2025b), we find that incorporating R_{tool} yields no performance gain over correctness-based rewards alone. We conjecture that penalizing execution errors encourages the model to generate overly simplistic code, thereby hindering its ability to solve problems correctly.

7 Conclusion

In this work, we presented J-ToRL, an end-to-end RL framework that trains LLM-based judges to invoke computational tools and to integrate tool outputs into their judgments. Experiments across benchmarks demonstrate that J-ToRL-trained judges improve over their base counterparts and achieve state-of-the-art performance among specialized judges while narrowing the gap to frontier judge models. Moreover, J-ToRL framework produces judges capable of monitoring, validating, and adaptively refining its tool usage strategies. These findings suggest that end-to-end RL tool integration is a promising path toward building judges for evaluations demanding precise computations.

Limitations

Our current framework primarily utilizes code interpreters; extending to a broader tool ecosystems such as retrieval, theorem provers, simulators and to multimodal evaluation would further increase its applicability. Further, although our training data spans diverse task types, the distribution may not fully capture the long-tail of real-world evaluation scenarios, particularly in open-ended creative or subjective assessment tasks where tool utility is less clear. Our current experiments explicitly focus on judgment tasks within reasoning domains, which inherently have clear, non-subjective reference answers. Nonetheless, our proposed framework is generalizable to more open-ended, subjective, or creative tasks. For instance, by leveraging rubric-based rewards as auxiliary training signals (Gunjal et al., 2026), structured rubrics can effectively decompose holistic judgments into multiple verifiable sub-criteria (e.g., grammatical correctness, coherence, stylistic adherence). This approach enriches the reward structure by providing a fine-grained, multi-dimensional feedback structure even in the absence of explicit gold standards. Importantly, such structured rubrics impose stricter evaluative criteria compared to accuracy on preference judgments alone, as models must concurrently meet multiple precise conditions. We hope J-ToRL serves as a step toward a scalable general evaluation systems that are more capable and trustworthy.

References

Maosong Cao, Alexander Lam, Haodong Duan, Hongwei Liu, Songyang Zhang, and Kai Chen. 2024. Compassjudger-1: All-in-one judge model helps model evaluation and evolution. *arXiv preprint arXiv:2410.16256*.

Xiusi Chen, Gaotang Li, Ziqi Wang, Bowen Jin, Cheng Qian, Yu Wang, Hongru Wang, Yu Zhang, Denghui Zhang, Tong Zhang, and 1 others. 2025. Rm-r1: Reward modeling as reasoning. *arXiv preprint arXiv:2505.02387*.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. 2025. Retool: Reinforcement learning for strategic tool use in llms. *arXiv preprint arXiv:2504.11536*.

Arduin Findeis, Floris Weers, Guoli Yin, Ke Ye, Ruoming Pang, and Tom Gunter. 2025. Can external validation tools improve annotation quality for llm-as-a-judge? In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15997–16020.

Evan Frick, Tianle Li, Connor Chen, Wei-Lin Chiang, Anastasios Nikolas Angelopoulos, Jiantao Jiao, Banghua Zhu, Joseph E. Gonzalez, and Ion Stoica. 2025. How to evaluate reward models for RLHF. In *The Thirteenth International Conference on Learning Representations*.

Jinlan Fu, See Kiong Ng, Zhengbao Jiang, and Pengfei Liu. 2024. Gptscore: Evaluate as you desire. In

Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pages 6556–6576.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Yunzhong He, Bing Liu, and Sean Hendryx. 2026. Rubrics as rewards: Reinforcement learning beyond verifiable domains. In *International Conference on Learning Representations*, volume 2026, pages 127924–127945.

Daya Guo and 1 others. 2025. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Wenfei Zhou, James Coady, David Peng, Yujie Qiao, Luke Benson, and 1 others. 2024. FOLIO: Natural language reasoning with first-order logic. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 22017–22031.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, and 1 others. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Seungone Kim, Jamin Shin, Yejin Cho, Joel Jang, Shayne Longpre, Hwaran Lee, Sangdoo Yun, Seongjin Shin, Sungdong Kim, James Thorne, and 1 others. 2023. Prometheus: Inducing fine-grained evaluation capability in language models. In *The Twelfth International Conference on Learning Representations*.

Neema Kotonya, Saran Krishnasamy, Joel Tetreault, and Alejandro Jaimes. 2023. Little giants: Exploring the potential of small llms as evaluation metrics in summarization in the eval4nlp 2023 shared task. In *Proceedings of the 4th Workshop on Evaluation and Comparison of NLP Systems*, pages 202–218.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, and 1 others. 2024. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*.

Dawei Li, Bohan Jiang, Liangjie Huang, Alimohammad Beigi, Chengshuai Zhao, Zhen Tan, Amrita Bhattacharjee, Yuxuan Jiang, Canyu Chen, Tianhao Wu, and 1 others. 2025a. From generation to judgment: Opportunities and challenges of llm-as-a-judge. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 2757–2791.

Junlong Li, Shichao Sun, Weizhe Yuan, Run-Ze Fan, Pengfei Liu, and 1 others. 2024a. Generative judge for evaluating alignment. In *The Twelfth International Conference on Learning Representations*.

Junlong Li, Shichao Sun, Weizhe Yuan, Run-Ze Fan, Hai Zhao, and Pengfei Liu. 2023a. Generative judge for evaluating alignment. *arXiv preprint arXiv:2310.05470*.

Lei Li, Yekun Chai, Shuohuan Wang, Yu Sun, Hao Tian, Ningyu Zhang, and Hua Wu. 2023b. Tool-augmented reward modeling. *arXiv preprint arXiv:2310.01045*.

Lei Li, Yekun Chai, Shuohuan Wang, Yu Sun, Hao Tian, Ningyu Zhang, and Hua Wu. 2024b. Tool-augmented reward modeling. In *The Twelfth International Conference on Learning Representations*.

Xuefeng Li, Haoyang Zou, and Pengfei Liu. 2025b. ToRL: Scaling tool-integrated RL. *arXiv preprint arXiv:2503.23383*.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, and 1 others. 2022. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097.

Yen-Ting Lin and Yun-Nung Chen. 2023. Llm-eval: Unified multi-dimensional automatic evaluation for open-domain conversations with large language models. In *Proceedings of the 5th Workshop on NLP for Conversational AI (NLP4ConvAI 2023)*, pages 47–58.

Yesheng Liu, Hao Li, Haiyu Xu, Baoqi Pei, Jiahao Wang, Mingxuan Zhao, Jingshu Zheng, Zheqi He, JG Yao, Bowen Qin, and 1 others. 2025a. Beyond multiple choice: Verifiable openqa for robust vision-language rft. *arXiv e-prints*, pages arXiv–2511.

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

Zeyao Ma, Xiaokang Zhang, Jing Zhang, Jifan Yu, Sijia Luo, and Jie Tang. 2025. Dynamic scaling of unit tests for code reward modeling. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6917–6935, Vienna, Austria. Association for Computational Linguistics.

Dakota Mahan, Duy Van Phung, Alon Albalak, Rafael Rafailov, Chase Blagden, Nathan Lile, Louis Castricato, Jan-Philipp Fränken, and Chelsea Finn. 2025. Generative reward models.

Seyed Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. 2025. GSM-symbolic: Understanding the limitations of mathematical reasoning in large language models. In *The Thirteenth International Conference on Learning Representations*.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, and 1 others. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.

OpenAI. 2024a. GPT-4o system card. *arXiv preprint arXiv:2410.21276*.

OpenAI. 2024b. Learning to reason with LLMs.

Hao Peng, Yunjia Qi, Xiaozhi Wang, Zijun Yao, Bin Xu, Lei Hou, and Juanzi Li. 2025. Agentic reward modeling: Integrating human preferences with verifiable correctness signals for reliable reward systems. *arXiv preprint arXiv:2502.19328*.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru WANG, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. 2025. ToolRL: Reward is all tool learning needs. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Shuofei Qiao, Honghao Gui, Chengfei Lv, Qianghuai Jia, Huajun Chen, and Ningyu Zhang. 2024. Making language models better tool learners with execution feedback. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3550–3568.

Vyas Raina, Adian Liusie, and Mark Gales. 2024. Is llm-as-a-judge robust? investigating universal adversarial attacks on zero-shot llm assessment. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 7499–7517.

Swarnadeep Saha, Xian Li, Marjan Ghazvininejad, Jason Weston, and Tianlu Wang. 2025. Learning to plan & reason for evaluation with thinking-llm-as-a-judge. *arXiv preprint arXiv:2501.18099*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551.

Amrith Setlur, Saurabh Garg, Xinyang Geng, Naman Garg, Virginia Smith, and Aviral Kumar. 2024. Rl on incorrect synthetic data scales the efficiency of llm math reasoning by eight-fold. *Advances in Neural Information Processing Systems*, 37:43000–43031.

Zhihong Shao and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

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

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

Sijun Tan, Siyuan Zhuang, Kyle Montgomery, William Y Tang, Alejandro Cuadron, Chenguang Wang, Raluca Ada Popa, and Ion Stoica. 2024. Judgebench: A benchmark for evaluating llm-based judges. *arXiv preprint arXiv:2410.12784*.

Qwen Team. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Tu Vu, Kalpesh Krishna, Salaheddin Alzubi, Chris Tar, Manaal Faruqui, and Yun-Hsuan Sung. 2024. Foundational autoraters: Taming large language models for better automatic evaluation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 17086–17105.

Tianlu Wang, Ilia Kulikov, Olga Golovneva, Ping Yu, Weizhe Yuan, Jane Dwivedi-Yu, Richard Yuanzhe Pang, Maryam Fazel-Zarandi, Jason Weston, and Xian Li. 2024. Self-taught evaluators. *arXiv preprint arXiv:2408.02666*.

Yidong Wang, Zhuohao Yu, Wenjin Yao, Zhengran Zeng, Linyi Yang, Cunxiang Wang, Hao Chen, Chaoya Jiang, Rui Xie, Jindong Wang, and 1 others. 2023. Pandalm: An automatic evaluation benchmark for llm instruction tuning optimization. In *The Twelfth International Conference on Learning Representations*.

Chenxi Whitehouse, Tianlu Wang, Ping Yu, Xian Li, Jason Weston, Ilia Kulikov, and Swarnadeep Saha. 2025. J1: Incentivizing thinking in llm-as-a-judge via reinforcement learning. *arXiv preprint arXiv:2505.10320*.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256.

Austin Xu, Yilun Zhou, Xuan-Phi Nguyen, Caiming Xiong, and Shafiq Joty. 2025a. J4r: Learning to judge with equivalent initial state group relative policy optimization. *arXiv preprint arXiv:2505.13346*.

Kaishuai Xu, Tiezheng Yu, Yi Cheng, Wenjun Hou, Liangyou Li, Xin Jiang, Lifeng Shang, Qun Liu, and Wenjie Li. 2025b. Learning to align multi-faceted evaluation: A unified and robust framework. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 9488–9502.

Zhenran Xu, Senbao Shi, Baotian Hu, Jindi Yu, Dongfang Li, Min Zhang, and Yuxiang Wu. 2023. Towards reasoning in large language models via multi-agent peer review collaboration. *arXiv preprint arXiv:2311.08152*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

Jiachen Yu, Shaoning Sun, Xiaohui Hu, Jiaxu Yan, Kaidong Yu, and Xuelong Li. 2025a. Improve LLM-as-a-judge ability as a general ability. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 14110–14126, Suzhou, China. Association for Computational Linguistics.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, and 1 others. 2025b. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*.

Weihao Yu, Zihang Jiang, Yanfei Dong, and Jiashi Feng. 2020. ReClor: A reading comprehension dataset requiring logical reasoning. In *International Conference on Learning Representations*.

Lifan Yuan, Ganqu Cui, Hanbin Wang, Ning Ding, Xingyao Wang, Jia Deng, Boji Shan, Huimin Chen, Ruobing Xie, Yankai Lin, Zhenghao Liu, Bowen Zhou, Hao Peng, Zhiyuan Liu, and Maosong Sun. 2024. Advancing llm reasoning generalists with preference trees. *Preprint*, arXiv:2404.02078.

Qiyuan Zhang, Yufei Wang, Yuxin Jiang, Chuhan Wu, Liangyou Li, Yasheng Wang, Xin Jiang, Lifeng Shang, Ruiming Tang, Fuyuan Lyu, and 1 others. 2024. Reviseval: Improving llm-as-a-judge via response-adapted references. In *The Thirteenth International Conference on Learning Representations*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623.

Aojun Zhou, Ke Wang, Zimu Lu, Weikang Shi, Sichun Luo, Zipeng Qin, Shaoqing Lu, Anya Jia, Linqi Song, Mingjie Zhan, and 1 others. 2023. Solving challenging math word problems using gpt-4 code interpreter with code-based self-verification. *arXiv preprint arXiv:2308.07921*.

Mingchen Zhuge, Changsheng Zhao, Dylan R Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, and 1 others. 2024. Agent-as-a-judge: Evaluate agents with agents. In *Forty-second International Conference on Machine Learning*.

Appendix

A Further Training Details

KV-cache reuse. We follow (Feng et al., 2025) implementation in verl to reduce rollout memory overhead, and cache all KV-cache computed before code execution. We avoid recomputation and only compute-append the KV-cache corresponding to interpreter feedback. This strategy substantially reduces KV-cache memory footprint across rollouts.

B Prompt Templates

We attach the prompt templates utilized in all of our experiments in Table 9, 10, and 11.

You are given a [User Question] and two responses from two AI assistants. Act as an impartial judge and evaluate which response better follows the user's instructions and provides a higher-quality answer. Let's think step by step.

Identify any flaws and weakness in logic, incorrect computations, or invalid assumptions in each response. Identify any incorrect or invalid reasoning steps within both of the responses. If the [User Question] is exact (have a single correct answer), you can first work through the problem to derive the correct reference answer and then compare each response against it to judge how well the responses satisfy the reference answer. If the [User Question] is relative (do not have one fixed answer), derive the evaluation criteria for a high-quality response and compare each response against them to judge how well the responses satisfy these criteria. Be explicit in your thought process on evaluating the reasoning and thinking behind each responses.

At the end of your output, write your final verdict on which assistant is better, by strictly following this format: $\boxed{A}$ if assistant A is better, $\boxed{B}$ if assistant B is better.

Table 9: J-ToRL prompt template for experiments without the coding tool integration.

You are given a [User Question] and two responses from two AI assistants. Act as an impartial judge and evaluate which response better follows the user's instructions and provides a higher-quality answer. **You have the ability to write executable Python code to enhance your reasoning process, especially when the [User Question] involves math or logic.** Let's think step by step.

Identify any flaws and weakness in logic, incorrect computations, or invalid assumptions in each response. **You can generate Python code to help** identify any incorrect or invalid reasoning steps within both of the responses. If the [User Question] is exact (have a single correct answer), you can first work through the problem to derive the correct reference answer and then compare each response against it to judge how well the responses satisfy the reference answer. **You can generate Python code to help derive the reference answer**. If the [User Question] is relative (do not have one fixed answer), derive the evaluation criteria for a high-quality response and compare each response against them to judge how well the responses satisfy these criteria. Be explicit in your thought process on evaluating the reasoning and thinking behind each responses.

At the end of your output, write your final verdict on which assistant is better, by strictly following this format: $\boxed{A}$ if assistant A is better, $\boxed{B}$ if assistant B is better.

Table 10: J-ToRL prompt template for experiments with coding tool integration.

You are given a [User Question] and two responses from two AI assistants. Act as an impartial judge and evaluate which response better follows the user's instructions and provides a higher-quality answer. You have the ability to write executable Python code to enhance your reasoning process, especially when the [User Question] involves math or logic. Let's think step by step.

Identify any flaws and weakness in logic, incorrect computations, or invalid assumptions in each response. You can generate Python code to help identify any incorrect or invalid reasoning steps within both of the responses. If the [User Question] is exact (have a single correct answer), you can first work through the problem to derive the correct reference answer and then compare each response against it to judge how well the responses satisfy the reference answer. You can generate Python code to help derive the reference answer. **If you derive the reference answer, state the reference answer by strictly following this format `<answer>reference_answer</answer>`.** If the [User Question] is relative (do not have one fixed answer), derive the evaluation criteria for a high-quality response and compare each response against them to judge how well the responses satisfy these criteria. Be explicit in your thought process on evaluating the reasoning and thinking behind each responses.

At the end of your output, write your final verdict by strictly following this format: $\boxed{A}$ if assistant A is better, $\boxed{B}$ if assistant B is better.

Table 11: J-ToRL prompt template for experiments with coding tool integration and reference answer rewards (R_{rans}).