JudgePanel: A Compact Judge with Panel Deliberation via Adaptive Multi-Reward Reinforcement Learning

Yiyue Qian*, **Shinan Zhang***, **Huan Song**, **Hannah Marlowe**
{iamyiyue, shinanz, huanso, marloweh}@amazon.com

AWS Generative AI Innovation Center, USA

Abstract

The LLM-as-a-Judge paradigm has emerged as a scalable alternative to human evaluation. However, single-model judges are limited by their inherent model biases, while multi-agent evaluation protocols that mitigate this through diverse deliberation are prohibitively expensive at inference time. To this end, we propose **JudgePanel**, which equips a compact Judge model with multi-agent Panel deliberation capability. Specifically, we first train on panel deliberation traces from an ensemble of strong evaluators, capturing structured patterns of discussion, disagreement, and resolution. To further improve judgment quality beyond SFT, we introduce *AdaReward*, an adaptive multi-reward RL algorithm that dynamically rebalances reward component weights as different objectives saturate at different rates during RL training. For practical deployment, we further design a lightweight domain specialization module for rapid adaptation to new evaluation domains with few hundred labeled samples. As a result, (i) *Novel*: the first framework to equip a single compact judge with multi-agent panel deliberation capability at single-model inference cost; (ii) *Effective & Reliable*: JudgePanel with a 14B backbone outperforms judge-specialized models up to 70B across four evaluation benchmarks, demonstrates strong position consistency, and rapidly specializes to new domains with few hundred samples.

1 Introduction

As large language models (LLMs) continue to advance, reliably evaluating their outputs has become increasingly challenging due to the high cost and limited scalability of human evaluation (Clark et al., 2021; Karpinska et al., 2021; Qian et al., 2025). The LLM-as-a-Judge paradigm, where LLMs generate chain-of-thought reasoning before producing a verdict, has emerged as a promising alternative (Zheng et al., 2023; Kim et al., 2024; Saha et al., 2024). However, single-model judges are inherently limited by their model biases stemming from training data and optimization, leading to systematic evaluation biases and inconsistent judgments across domains (Chen et al., 2024; Wang et al., 2024b).

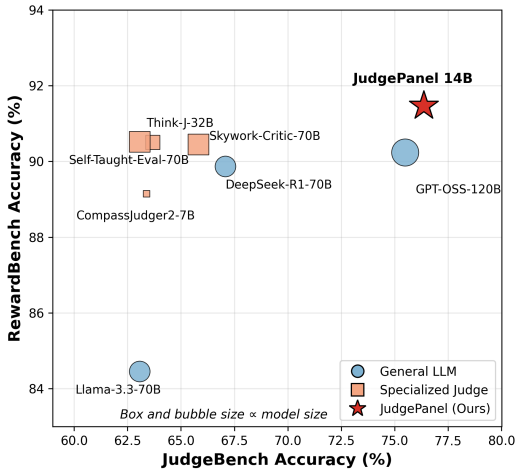

Figure 1: JudgeBench vs. RewardBench accuracy. Existing models show inconsistent cross-dataset performance, while JudgePanel achieves the best on both.

To mitigate these biases, recent work employs multi-agent evaluation protocols where multiple LLMs assess, discuss disagreements, and converge on a consensus (Qian et al., 2026b; Chan et al., 2024; Saha et al., 2024). While effective at reducing individual model biases through diverse perspectives, such protocols require deploying multiple large models with multi-round inference, making them prohibitively expensive for large-scale deployment. This raises a natural question: *can we train a single compact model that reasons like a panel of experts?*

To this end, we propose **JudgePanel**, which equips a compact Judge model with multi-agent Panel deliberation capability. We first construct panel deliberation traces, capturing structured pat-

* Equal contribution.

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 19253–19267
October 24-29, 2026 ©2026 Association for Computational Linguistics

terns of discussion, disagreement, and resolution. By training a compact model on these traces via SFT, we equip it with panel deliberation at single-model inference cost. However, SFT relies on pattern imitation rather than deep reasoning (Chen et al., 2025), limiting the model to reproducing the quality of its training traces without the ability to generalize beyond them. To address this, we apply multi-reward RL that jointly optimizes judgment quality from multiple perspectives, such as format compliance, discussion quality, and outcome correctness. Yet in multi-reward training, different components learn at different rates: format compliance saturates early while outcome correctness requires sustained optimization, and fixed weights cannot capture this dynamic. We therefore propose **AdaReward**, an adaptive weight learner that dynamically rebalances reward components during RL training, creating a curriculum-like effect without manual tuning. Furthermore, as illustrated in Figure 1, even well-trained judges show inconsistent performance across evaluation domains. Specialized judge models excel on some benchmarks while underperforming on others. For practical deployment, we design a lightweight domain specialization module that rapidly adapts the judge to new domains with a few hundred labeled samples.

We summarize our contributions as follows:

- **Novel Framework**: We propose JudgePanel, the first to equip a single compact judge with panel deliberation capability via adaptive multi-reward RL, achieving multi-agent reasoning quality at single-model inference cost.
- **AdaReward**: We introduce an adaptive multi-reward RL algorithm that dynamically rebalances reward component weights during RL training, shifting focus from saturated components to those still providing useful signals.
- **State-of-the-Art Performance**: JudgePanel with a 14B backbone outperforms larger judge-specialized models (i.e., 70B) across four evaluation benchmarks, demonstrates strong position consistency, and rapidly specializes to new domains with few hundred samples.

2 Preliminary

LLM-as-a-Judge. Given an evaluation input x_i (e.g., an instruction with candidate responses), an LLM-as-a-Judge π_θ generates a reasoning trace t_i followed by a verdict $\hat{y}_i$. During RL training, we sample G rollouts $\{\tau_{i,j}\}_{j=1}^{G}$ from the reference policy π_{ref}, where each rollout $\tau_{i,j} = (t_{i,j}, \hat{y}_{i,j})$ contains a reasoning trace and predicted verdict.

Group reward-Decoupled Normalization Policy Optimization (GDPO). To optimize judge with K reward functions $\{R_k\}_{k=1}^{K}$, inspired by GDPO (Liu et al., 2026a), which resolves the *reward signal collapse* of standard Group Relative Policy Optimization (GRPO) (Shao et al., 2024) by normalizing each reward before aggregation, we employ GDPO as our base RL algorithm. GDPO computes per-reward advantages $A_k^{(i,j)} = (R_k(\tau_{i,j}) - \text{mean}_j\{R_k\})/\text{std}_j\{R_k\}$ and aggregates them as:

$$A_{\text{sum}}^{(i,j)} = \sum_{k=1}^{K} w_k \cdot A_k^{(i,j)},$$
$$\hat{A}_{\text{sum}}^{(i,j)} = \frac{A_{\text{sum}}^{(i,j)} - \text{mean}_{\text{batch}}\{A_{\text{sum}}\}}{\text{std}_{\text{batch}}\{A_{\text{sum}}\}}, \quad (1)$$

with fixed weights $\{w_k\}$. The policy is optimized via a clipped surrogate objective $\mathcal{J}_{\text{GDPO}}(\theta)$.

3 JudgePanel: Panel Deliberation via Adaptive Multi-Reward RL

As illustrated in Figure 2, JudgePanel is a three-stage framework: (i) *Panel Deliberation*, which generates collaborative evaluation traces for model training; (ii) *AdaReward*, an adaptive reward weight learner for multi-reward RL training; and (iii) *Lightweight Domain Specialization*, which specializes the judge to new domains with hundred samples via LoRA training.

3.1 Panel Deliberation via Multi-Agent Collaboration

Existing LLM-as-a-Judge approaches (Whitehouse et al., 2026; Huang et al., 2026; Saha et al., 2025; Zhang et al., 2025) rely on a single model to generate synthetic training data, creating a reasoning diversity bottleneck that misses the multi-model collaborative deliberation characteristic of high-quality human evaluation. To overcome this, following the similar mechanism of CollabEval (Qian et al., 2026b), we obtain the reasoning traces samples from Chatbot Arena dataset (Chiang et al., 2024), a large-scale collection of real user conversations with deployed LLMs.

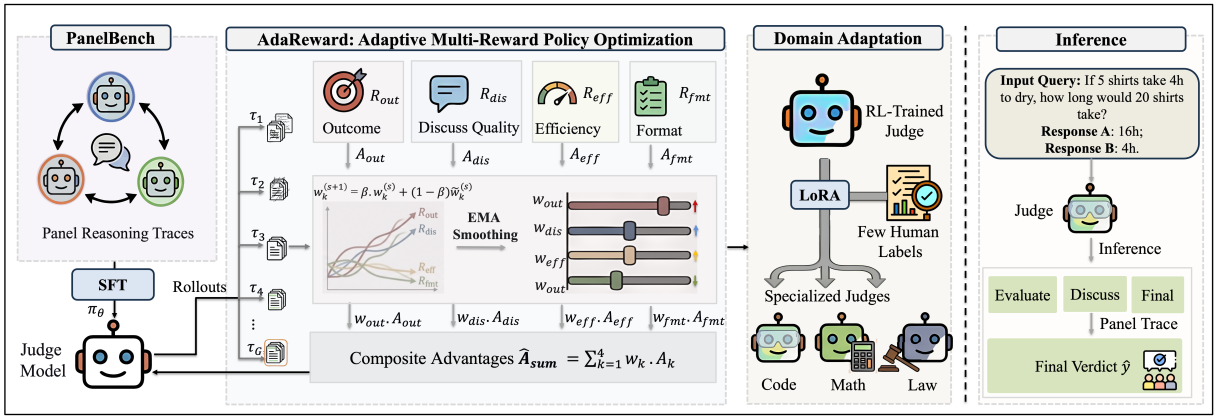

Figure 2: The overall framework of JudgePanel: (a) *PanelBench*: A panel evaluation through multi-round discussion to produce panel deliberation traces for training a judge model π_θ via SFT. (b) *AdaReward*: The judge generates rollouts scored by four reward components. An EMA-based adaptive weight learner dynamically rebalances reward weights, producing composite advantages for policy updates. (c) *Domain Adaptation*: The RL-trained judge is specialized to new domains via LoRA fine-tuning on a few hundred human-labeled samples. (d) *Inference*: At deployment, the compact judge generates a full panel deliberation trace at single-model inference cost.

Input

Instruction: If 5 shirts take 4h to dry in the sun, how long would 20 shirts take?
Response A: Proportion: $5/4 = 20/x \Rightarrow x = 16$h.
Response B: Per-shirt: $4/5 = 0.8$h $\times 20 = 16$h.

Phase 1: Independent Evaluation

Speaker 1: B is clearer. **Verdict: b** | Conf: 5
Speaker 2: Both wrong—drying is parallel (4h). **Verdict: tie** | Conf: 4
Speaker 3: Both equally good under linear assumption. **Verdict: tie** | Conf: 5
→ *No consensus. Proceeding to discussion.*

Phase 2: Discussion

Speaker 1: Acknowledges parallel point, lowers confidence. **Revised: b** | Conf: 4
Speaker 2: Both equally wrong; clarity gap insufficient. **Verdict: tie** | Conf: 4
Speaker 3: Agrees with Speaker 2, B's edge is minimal. **Verdict: tie** | Conf: 5
→ *No consensus. Proceeding to Final Judge.*

Phase 3: Final Judge

Drying is parallel—correct answer is 4h. Both responses apply linear scaling incorrectly. Speaker 2's identification of the parallel-process error is most analytically sound. Stylistic difference is insufficient.
Final Verdict: tie | Conf: 4

Output (Training Target)

t_i = [Initial Eval: Speaker 1, 2, 3] → [Discussion: Speaker 1, 2, 3] → [Final Judge] → **Verdict: tie**

Figure 3: Simplified three-stage panel reasoning trace.

As illustrated in Figure 3, the panel follows a three-phase protocol: (i) ***Independent Evaluation***: each agent independently produces a structured evaluation $e_i^{(m)}$, verdict $v_i^{(m)}$, and confidence $c_i^{(m)}$; (ii) ***Multi-Round Discussion***: triggered when no consensus is reached, evaluators review the full history and provide updated assessments with explicit agreements, disagreements, and revised reasoning; and (iii) ***Final Judgment***: when discussion fails to converge or reaches $d_{\max}$ rounds, a final judge $\pi^{(\text{final})}$ analyzes the complete history and produces the verdict $\hat{y}_i$. We retain only samples where the panel verdict matches the ground-truth label y_i, yielding ∼8,700 correct panel traces for training. By training on these collaborative traces, the compact judge model (i.e., 14B) acquires panel deliberation capabilities, which learns to generate independent evaluation, disagreement identification, deliberation, and consensus.

3.2 AdaReward: Adaptive Multi-Reward Policy Optimization

As established in Section 2, GDPO resolves the reward signal collapse problem inherent in GRPO's multi-reward optimization by decoupling per-reward normalization. However, GDPO combines the normalized advantages with fixed weights $\{w_k\}$ in Equation 1 that remain constant throughout training. In judge training, we observe that reward components exhibit vastly different convergence dynamics. For instance, format compliance saturates quickly at around 50 steps, while outcome correctness, discussion quality, and efficiency are learning throughout training. To this end, we propose **AdaReward**, an adaptive weight learner

that dynamically learns reward component weights based on their training progress.

3.2.1 Multi-Aspect Reward Design

We design $K = 4$ reward functions that jointly capture both judgment correctness and reasoning quality. Each reward $R_k(\tau_{i,j})$ evaluates an aspect of the panel deliberation trace $\tau_{i,j}$. The detailed scoring of each reward is described in Appendix A.

Fine-Grained Outcome Reward R_{out}. Rather than a binary correct/incorrect reward, we design a graded outcome reward that diagnoses *where the reasoning went wrong*: whether the error originated in the initial evaluation, was introduced during discussion, or occurred at the final judgment stage. It provides richer signal than binary rewards.

Discussion Quality Reward R_{dis}. This reward measures whether the multi-round discussion produces genuine deliberation and whether speakers actually revise their positions based on others' arguments, rather than merely restating their evaluations. Crucially, this reward evaluates the quality of the deliberation process itself, rewarding genuine position changes that lead to correct outcomes rather than cosmetic flips.

Efficiency Reward R_{eff}. This reward evaluates whether the model's decision to engage in or skip discussion was beneficial, rewarding efficient consensus and penalizing harmful deliberation. R_{eff} explicitly conditions on correctness: the strongest penalty targets cases where discussion flipped a correct initial majority to a wrong final verdict, directly discouraging counterproductive deliberation.

Format Compliance Reward R_{fmt}. This reward ensures the generated trace follows the multi-agent collaboration structure (required section tags, M speakers with complete evaluation fields, proper discussion structure), which is essential for parsing and calculating the other rewards correctly.

3.2.2 Adaptive Weighting via EMA Learner

With the four reward components defined, we now address the core limitation of fixed-weight aggregation in existing RL methods. In GDPO (Equation 1), the composite advantage is $A_{\text{sum}}^{(i,j)} = \sum_{k=1}^{K} w_k \cdot A_k^{(i,j)}$ with constant w_k. When a component k saturates (i.e., $V_k \triangleq \mathbb{E}[(A_k^{(i,j)})^2] \approx 0$), all rollouts receive nearly identical rewards for that component, providing no useful learning signal after GDPO's per-reward normalization. With fixed weights, when a component saturates, (e.g., R_{dis}), its contribution to the composite advantage vanishes, but its weight cannot be redistributed to harder components (e.g., R_{out} and R_{eff}) that still provide learning signals.

EMA-Based Adaptive Weight Learner. Our key idea is to adjust each weight proportionally to how far its reward is from a target score, pushing harder on lagging components and easing off on saturated ones. Specifically, we define an gap signal $\delta_k^{(s)} = \rho_k - \bar{R}_k^{(s)}$ measuring the gap between the target ρ_k and the current batch-average reward $\bar{R}_k^{(s)}$ for component k at step s. The weight update uses *asymmetric scaling*: when a component falls behind its target ($\delta_k > 0$), its weight increases aggressively; when it exceeds the target ($\delta_k \leq 0$), the weight decreases conservatively. Concretely, we first compute a target weight $\tilde{w}_k^{(s)}$:

$$\tilde{w}_k^{(s)} = w_k^{(s)} \cdot \begin{cases} 1 + \alpha^+ \cdot \delta_k^{(s)} & \text{if } \delta_k^{(s)} > 0 \ \text{(below target)} \\ 1 + \alpha^- \cdot \delta_k^{(s)} & \text{if } \delta_k^{(s)} \leq 0 \ \text{(above target)}, \end{cases} \tag{2}$$

where $\alpha^+ > \alpha^-$ ensures faster recovery for underperforming components than decay for well-performing ones. The target weight is then clipped to $[w_k^{\min}, w_k^{\max}]$ before updates (Eq. 3), ensuring no single component dominates or vanishes regardless of the gap magnitude. The final weight is updated via exponential moving average (EMA):

$$w_k^{(s+1)} = \beta \cdot w_k^{(s)} + (1 - \beta) \cdot \tilde{w}_k^{(s)}, \tag{3}$$

where $\beta \in (0, 1)$ controls the adaptation speed (lower β means faster response to new targets). Weights are updated every Δ steps, balancing responsiveness (frequent updates) against stability (allowing the policy to learn from current weights before rebalancing).

Intuition. Fixed weights apply constant correction regardless of how far each reward is from its target, while AdaReward's correction magnitude scales with the gap δ_k, pushing harder when far from target and easing off when close. This creates a curriculum-like effect: as easy components (e.g., R_{fmt}) reach their targets, the learner reallocates weight to harder components (e.g., R_{out}) where δ_k remains large.

Furthermore, Table 3(f) shows that adaptive weights outperforms fixed weights.

Full AdaReward Objective. Combining the decoupled normalization with the adaptive weights, the AdaReward objective is formulated as:

$$\mathcal{J}_{\text{Ada}}(\theta) = \mathbb{E}_{x_i \sim \mathcal{D},\, \{\tau_{i,j}\}_{j=1}^{G} \sim \pi_{\text{ref}}} \Bigg[\frac{1}{G} \sum_{j=1}^{G} \frac{1}{|\tau_{i,j}|} \sum_{l=1}^{|\tau_{i,j}|} \min\Big(r_l(\theta)\, \hat{A}_{\text{sum}}^{(i,j,s)},\ \text{clip}(r_l(\theta), 1-\varepsilon, 1+\varepsilon)\, \hat{A}_{\text{sum}}^{(i,j,s)} \Big) \Bigg], \tag{4}$$

where $r_l(\theta) = \frac{\pi_\theta(\tau_{i,j}^{l} | x_i, \tau_{i,j}^{<l})}{\pi_{\text{ref}}(\tau_{i,j}^{l} | x_i, \tau_{i,j}^{<l})}$ is the importance sampling ratio at token position l. $\hat{A}_{\text{sum}}^{(i,j,s)}$ is the batch-normalized composite advantage computed using the time-varying weights $\{w_k^{(s)}\}$ at step s.

3.3 Lightweight Domain Specialization

As shown in Figure 1, even well-trained judges exhibit inconsistent performance across different evaluation domains. These judge-specialized models trained on narrow distributions struggle to generalize. For instance, Skywork-Critic-70B (Liu et al., 2026b) performs well on RewardBench but poorly on JudgeBench. In real-world deployment, different use cases demand different domain expertise, yet existing approaches do not provide a mechanism for rapid specialization. This motivates a lightweight mechanism for rapid domain specialization without retraining from scratch.

Our key insight is that a practitioner can quickly curate a small set of human-labeled samples (e.g., 200–500) for a new evaluation domain, then apply the panel protocol (Section 3.1) to automatically generate domain-specific panel deliberation traces from these labels.

We further fine-tune the RL-trained judge using LoRA (Hu et al., 2021) on these traces. LoRA is chosen over full fine-tuning to preserve general deliberation capabilities acquired during RL while efficiently injecting domain-specific knowledge with minimal parameters.

The full training pipeline of JudgePanel is summarized in Appendix Algorithm 1, and all hyperparameters settings are provided in Appendix C.1.

4 Experiments

4.1 Datasets and Baseline Models

PanelBench. Following the panel protocol (Section 3.1), we construct *PanelBench* from Chatbot Arena (Chiang et al., 2024) (20K pairwise samples with human labels). After filtering out incorrect verdicts, we retain 8,500 panel deliberation traces for SFT and RL training. Mention that, PanelBench contains no overlapping input queries or responses with any evaluation benchmark test set or domain adaptation data.

Benchmarks. We evaluate on four established judge benchmarks (details in Appendix C.2): (1) *JudgeBench* (JB) (Tan et al., 2025): 620 challenging pairs across knowledge, reasoning, math, and coding; (2) *RewardBench* (RB) (Lambert et al., 2025): 2,985 prompt-chosen-rejected trios; (3) *RM-Bench* (RM) (Liu et al., 2025a): 3,981 samples testing sensitivity to subtle content differences; (4) *PPE* (Frick et al., 2025): 16,038 multilingual pairwise preference samples.

Domain Specialization Data. To simulate real-world deployment, we generate DA data from related but independent datasets for each benchmark. As some DA source datasets only provide ground-truth labels without preference pairs, we use Qwen2.5-72B-Instruct (Yang et al., 2025) to generate incorrect responses, constructing pairwise preference samples. We then run our panel protocol on these pairs, retaining 200–500 correctly predicted samples per benchmark for LoRA adaptation (details in Appendix C.2).

Baselines. We compare against (1) *General LLMs*: Qwen2.5-7B-Instruct (Yang et al., 2025), DeepSeek-R1-8B (Guo et al., 2025), Llama-3.3-70B (Grattafiori et al., 2024), DeepSeek-R1-Distill-Llama-70B (Shao et al., 2024); and (2) *Judge-specialized models*: JudgeLRM-3B/7B^{RL} (Chen et al., 2025), CompassJudger2-7B$^{\text{SFT+RL}}$ (Zhang et al., 2025), ArmoRM-8B^{SFT} (Wang et al., 2024a), DeepSeek-GRM-27B$^{\text{SFT+RL}}$ (Yang et al., 2024), Think-J-32B$^{\text{SFT+RL}}$ (Huang et al., 2026), Self-Taught-Eval-70B^{SFT} (Wang et al., 2024c), Skywork-Critic-70B^{SFT} (Liu et al., 2026b).

Our Models. We train JudgePanel using Qwen3-14B (Yang et al., 2025) with three stages: SFT on PanelBench, AdaReward RL, and LoRA domain adaptation. For fair comparison, we provide all baseline and our model parsed accuracy as the performance results in this paper. The detailed **hyperparameters** for each stage are **in Appendix C.1**.

4.2 Main Results

Effectiveness. Table 1 reports the accuracy performance on four benchmarks. We highlight the following key findings: (i) Judge-specialized models do not consistently outperform general LLMs; Even large judge-specialized models such as Skywork-Critic-70B and Self-Taught-Eval-70B fail to maintain consistently strong performance

Table 1: Accuracy performance (%) comparison among general LLMs and judge-specialized models on JudgeBench, RewardBench, RMBench, and PPE. Avg. Rank is the mean rank across all four benchmarks (lower is better). The bolded numbers are the best results and the underlined numbers indicate the runner-up results.

Type	Model	JudgeBench	RewardBench	RMBench	PPE	Avg. Rank↓
General LLM	Qwen2.5-7B-Instruct	57.66	78.81	65.69	38.14	11.0
	DeepSeek-R1-8B	56.56	76.25	74.97	37.89	10.2
	Llama-3.3-70B	63.06	84.46	73.55	41.67	7.6
	DeepSeek-R1-70B	67.08	89.87	85.17	42.89	3.8
Specialized Judge	JudgeLRM-3B [RL]	60.00	64.09	58.58	36.00	12.8
	JudgeLRM-7B [RL]	55.84	76.28	66.54	37.80	12.2
	CompassJudger2-7B [SFT+RL]	63.39	89.15	75.33	40.26	6.2
	ArmoRM-8B [SFT]	55.81	88.44	69.23	37.26	11.2
	DeepSeek-GRM-27B [SFT+RL]	56.46	79.19	69.19	38.31	10.5
	Think-J-32B [SFT+RL]	63.68	90.50	79.80	40.15	5.2
	Self-Taught-Eval-70B [SFT]	63.06	90.52	74.64	41.71	5.4
	Skywork-Critic-70B [SFT]	65.81	90.45	74.50	40.49	5.8
Ours	Qwen3-14B (main)	74.45	91.33	87.54	45.61	2.0
	Qwen3-14B (w/ DA)	**77.93**	**93.33**	**89.99**	**46.69**	**1.0**

Table 2: Position consistency and stable accuracy analysis (selected models). Cons. (%,↑): percentage of samples where verdicts correctly track swaps. Stable Acc. (%,↑): percentage of samples correctly predicted in *both* orderings.

Type	Model	JudgeBench		RewardBench		RMBench		PPE	
		Cons.	Stable Acc.	Cons.	Stable Acc.	Cons.	Stable Acc.	Cons.	Stable Acc.
General LLM	Llama-3.3-70B	68.87	46.94	88.51	79.77	77.24	62.35	74.45	34.78
	DeepSeek-R1-70B	71.95	52.97	91.73	85.86	86.82	79.05	73.57	34.66
Specialized Judge	CompassJudger2-7B [SFT+RL]	75.16	49.68	93.53	88.94	78.60	65.11	72.41	34.93
	Self-Taught-Eval-70B [SFT]	70.48	47.74	89.71	85.52	76.82	62.53	74.16	33.87
	Skywork-Critic-70B [SFT]	79.35	54.68	92.73	88.94	80.83	65.54	71.43	35.13
Ours	Qwen3-14B (main)	82.61	67.98	92.35	89.21	89.82	89.54	75.32	35.45
	Qwen3-14B (w/ DA)	**83.61**	**69.61**	**94.82**	**90.00**	**90.46**	**89.78**	**77.30**	**36.62**

across all benchmarks, confirming the cross-domain generalization challenge identified in Figure 1. (ii) RL-only judges underperform SFT-based judges at comparable sizes, indicating RL alone without quality SFT initialization is insufficient. (iii) JudgePanel with domain adaptation achieves the highest average rank, outperforming all baseline models with much bigger size. (iv) Without DA, JudgePanel already surpasses all baselines and judge-specialized models. We note that domain adaptation is a general technique applicable to any model; we report both settings for completeness. We would also discuss the effects of domain adaption samples on the model performance later.

Reliability. Table 2 presents position consistency and stable accuracy analysis on selected strong baselines. We can found out most judge-specialized models suffer from positional bias. JudgePanel achieves the highest consistency and stable accuracy on all benchmarks, surpassing all 70B models despite being 5 × smaller. This confirms that panel deliberation, which requires evaluating from multiple perspectives before reaching a verdict, encourages content-grounded evaluation that is robust to input ordering.

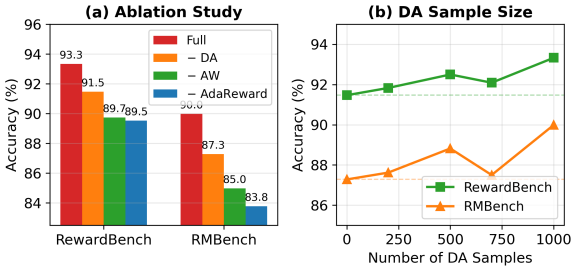

Figure 4: (a) Progressive ablation on RewardBench and RMBench. (b) DA sample size analysis; dashed lines indicate JudgePanel w/o DA.

4.3 Ablation and Analysis

Component Ablation and Reward Ablation Study. To validate each component, Figure 4(a)

Table 3: JudgePanel analysis. The highlighted row in gray is our method. RB = RewardBench, RM = RMBench.

(a) Generated Panel Trace Quality.

Metric	RB	RM
Constructive Change (%,↑)	91.0	85.7
Jaccard Distance (↑)	0.629	0.639
4-gram Overlap (%,↓)	14.4	15.5
Avg. Rounds	1.71	1.64

(b) Multi-Agent Protocol vs. JudgePanel vs. Think-J.

	Multi-Agent	JudgePanel	Think-J
RewardBench	94.54	91.33	90.50
PPE	46.28	45.61	40.15
Parameters	260B	14B	32B
Latency (n=100)	2.9 min	1.5 min	1.0 min

(c) Label Only vs. Single COT vs. Voting vs. Panel Trace.

Training Data	RB	RM
Label Only (SFT)	87.84	76.26
Single COT (SFT)	88.14	77.64
Majority Voting (SFT)	*88.50*	*78.56*
Panel Trace (Our SFT)	89.53	83.77

(d) Backbone Agnostic.

Model	RB	RM
Qwen3-32B (base)	88.21	74.52
Qwen3-32B (JudgePanel)	92.23	89.24
Qwen3-14B (base)	86.24	72.25
Qwen3-14B (JudgePanel)	91.33	87.54

(e) Reward Component Ablation.

Reward Setting	RB	RM
w/o Discussion Quality	88.24	84.74
w/o Efficiency	88.64	85.16
w/o Format Compliance	89.01	85.58
Full (ours)	91.33	87.54

(f) GDPO vs. GRPO.

RL Strategy	RB	RM
GRPO (fixed weights)	87.45	83.98
GRPO (adaptive weights)	89.98	85.64
GDPO (fixed weights)	89.74	84.98
GDPO (adaptive, ours)	91.33	87.54

shows progressive ablation on RewardBench and RMBench. Removing DA (−DA) drops performance by 1.86/2.71, confirming domain specialization is beneficial. Further replacing adaptive weights with fixed weights (−AW) causes an additional 1.73/2.30 drop, and removing RL entirely (−AdaReward, SFT only) widens the gap to 3.80/6.22 below the full system. Table 3(e) further shows that each reward component is essential: removing Discussion Quality (−3.09/−2.8) or Efficiency (−2.69/−2.38) each causes notable degradation, and removing Format Compliance also hurts, confirming that all four rewards capture distinct aspects of panel deliberation quality. To further verify the effectiveness of AdaReward, Table 3(f) shows our adaptive reward module leads to better final accuracy: GDPO with adaptive weights outperforms both GRPO and GDPO with fixed weights.

Panel Deliberation Analysis. To examine whether JudgePanel genuinely deliberates rather than mimicking format, Table 3(a) measures four metrics on RewardBench and RMBench: Constructive Change Rate (91.0%/85.7%) is far above the ∼50% random baseline, showing vote changes lead to correct answers; Jaccard Distance (0.629/0.639) confirms diverse reasoning across speakers; 4-gram Overlap (14.4%/15.5%) indicates discussion introduces novel content rather than repeating initial evaluations; and Average Rounds (1.71/1.64) shows the model adaptively engages in deliberation when needed (detailed metric definitions in Appendix C.3).

Beyond quality, we compare against the multi-agent protocol (CollabEval (Qian et al., 2026b)) as the quality ceiling and Think-J as a representative COT baseline. Table 3(b) shows JudgePanel (14B) achieves 91.33/45.61 on RB/PPE, approaching the multi-agent protocol (94.54/46.28) at half the latency (1.5 vs. 2.9 min for 100 samples). Think-J-32B achieves lower latency (1.0 min) but with lower accuracy (90.50/40.15), demonstrating that panel deliberation provides a better accuracy-efficiency tradeoff than single-model COT.

To isolate the data contribution, Table 3(c) compares four SFT strategies on the same Qwen3-14B backbone: Panel Trace SFT outperforms Label Only, Single COT, and Majority Voting (independent evaluations without discussion), demonstrating that multi-round deliberation provides richer supervision than either single-model reasoning or simple multi-perspective aggregation (qualitative examples in Appendix D).

Generalizability. To verify JudgePanel is not tied to a specific backbone, Table 3(d) shows applying it to Qwen3-32B yields +4.02/+14.72 on RB/RM, comparable to Qwen3-14B gains (+5.09/+15.32), confirming practitioners can select the backbone that fits their compute budget. For domain specialization, Figure 4(b) shows even 200 samples yield gains over the no-DA baseline, with diminishing returns beyond 500, making it practical for deployment where large-scale annotation is costly.

5 Related Work

LLM-as-a-Judge. As LLM is becoming increasingly powerful, more LLM-related applications are emerging (Xue et al., 2026; Ma et al., 2026b,a; Song et al., 2025; Ma et al., 2025). LLM-as-a-Judge, employing LLMs as automated evaluators (Zheng et al., 2023; Kim et al., 2024), has been widely used recently in these applications (Qian et al., 2026a; Li et al., 2025). However, general-purpose LLMs suffer from system-

atic biases (Chen et al., 2024; Wang et al., 2024b), motivating dedicated judge training. SFT-based approaches such as Self-Taught Evaluators (Wang et al., 2024c), Skywork-Critic (Liu et al., 2026b), ArmoRM (Wang et al., 2024a), and DeepSeek-GRM (Liu et al., 2025b) train judges through synthetic data curation, but rely on pattern memorization rather than flexible reasoning (Chen et al., 2025). Among RL-based approaches, JudgeLRM (Chen et al., 2025) demonstrates that judgment is inherently reasoning-intensive; Think-J (Huang et al., 2026) proposes judgment thinking initialization followed by GRPO-based optimization; CompassJudger-2 (Zhang et al., 2025) combines task-driven data curation with verifiable rewards; and J1 (Whitehouse et al., 2026) introduces consistency-based rewards both response orders, though neither code nor checkpoints have been publicly released. CollabEval (Qian et al., 2026b) demonstrates that multi-agent collaboration improves evaluation quality but requires deploying multiple large models at inference time. Inspired by this, we aim to train a compact model that internalizes multi-agent deliberation capability via adaptive multi-reward RL, eliminating the cost of multi-model deployment.

Multi-Reward RL Optimization. Recent LLM training increasingly incorporates multiple reward signals, such as correctness + format compliance (Chen et al., 2025). GDPO (Liu et al., 2026a) identifies that GRPO suffers from reward signal collapse in multi-reward settings and proposes decoupled per-reward normalization. Lu et al. (Lu et al., 2026) propose dynamic reward weighting using hypervolume-guided adaptation and gradient-based weight optimization to explore Pareto fronts. However, these approaches either use fixed weights (Aggarwal and Welleck, 2025; Chen et al., 2025), require manual reward function conditioning (Liu et al., 2026a), or need expensive per-objective gradient computation (Lu et al., 2026). Our AdaReward uses a lightweight learner that requires only scalar reward statistics, making it computationally cheap while achieving adaptive behavior.

6 Conclusion

In this work, we propose JudgePanel, a novel framework that equips a single compact judge model with multi-agent panel deliberation capability via adaptive multi-reward RL. Through extensive experiments on four established judge benchmarks, we demonstrate that JudgePanel (14B) consistently outperforms judge-specialized models (up to 70B), while maintaining strong position consistency. Furthermore, it also enables rapid specialization to new evaluation tasks with few hundred samples, demonstrating its practicality for real-world deployment. We believe JudgePanel provides an effective and scalable solution for practitioners who need reliable, domain-specific evaluation capabilities without the overhead of deploying large-scale models or multi-agent systems in production.

Limitations

Inference Token Cost. Due to the multi-agent reasoning trace, JudgePanel generates more tokens per evaluation than standard COT-based judges. The panel structure (independent evaluation round, discussion rounds, and final judgment) inherently requires longer outputs to simulate multi-speaker deliberation. However, as shown in Table 3(b), JudgePanel still achieves competitive latency (1.5 min vs. 1.0 min for Think-J on 100 samples) while outperforming COT-based Think-J in accuracy (91.33 vs. 90.50 on RewardBench, 45.61 vs. 40.15 on PPE) with a smaller model (14B vs. 32B). The accuracy gains justify the additional token cost for applications where judgment quality is prioritized over throughput.

Domain Specialization Pipeline. While JudgePanel can quickly adapt to new domains with as few as 200 labeled samples, the current domain specialization pipeline still requires running the panel protocol on these labeled samples to generate reasoning traces for LoRA fine-tuning. This introduces a dependency on the panel models during the specialization phase. Exploring more efficient domain specialization strategies, such as directly fine-tuning on verdict labels without trace generation, applying few-shot learning to the trained JudgePanel, or using the trained JudgePanel itself to generate specialization traces, is a promising direction for future research.

References

Pranjal Aggarwal and Sean Welleck. 2025. L1: Controlling how long a reasoning model thinks with reinforcement learning. *arXiv preprint arXiv:2503.04697*.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. 2024. Chateval: Towards better llm-based evaluators through multi-agent debate. In *ICLR*, volume 2024, pages 9079–9093.

Guiming Hardy Chen, Shunian Chen, Ziche Liu, Feng Jiang, and Benyou Wang. 2024. Humans or llms as the judge? a study on judgement bias. In *EMNLP*, pages 8301–8327.

Nuo Chen, Zhiyuan Hu, Qingyun Zou, Jiaying Wu, Qian Wang, Bryan Hooi, and Bingsheng He. 2025. Judgelrm: Large reasoning models as a judge. *arXiv preprint arXiv:2504.00050*.

Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios N Angelopoulos, Tianle Li, Dacheng Li, Banghua Zhu, Hao Zhang, Michael I Jordan, Joseph E Gonzalez, and 1 others. 2024. Chatbot arena: an open platform for evaluating llms by human preference. In *ICML*, pages 8359–8388.

Elizabeth Clark, Tal August, Sofia Serrano, Nikita Haduong, Suchin Gururangan, and Noah A Smith. 2021. All that's 'human'is not gold: Evaluating human evaluation of generated text. In *ACL-IJCNLP*, pages 7282–7296.

Evan Frick, Tianle Li, Connor Chen, Wei-Lin Chiang, Anastasios Angelopoulos, Jiantao Jiao, Banghua Zhu, Joseph E Gonzalez, and Ion Stoica. 2025. How to evaluate reward models for rlhf. In *ICLR*, volume 2025, pages 18128–18163.

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*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Hui Huang, Yancheng He, Hongli Zhou, Rui Zhang, Wei Liu, Weixun Wang, Jiaheng Liu, and Wenbo Su. 2026. Think-j: Learning to think for generative llm-as-a-judge. In *AAAI*, volume 40, pages 31158–31166.

Marzena Karpinska, Nader Akoury, and Mohit Iyyer. 2021. The perils of using mechanical turk to evaluate open-ended text generation. In *EMNLP*, pages 1265–1285.

Seungone Kim, Jay Shin, Joel Jang, Shayne Longpre, Hwaran Lee, Sangdoo Yun, Ryan Shin, Sungdong Kim, James Thorne, Minjoon Seo, and 1 others. 2024. Prometheus: Inducing fine-grained evaluation capability in language models. In *ICLR*, volume 2024, pages 29927–29962.

Nathan Lambert, Valentina Pyatkin, Jacob Morrison, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, and 1 others. 2025. Rewardbench: Evaluating reward models for language modeling. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 1755–1797.

Henger Li, Shuangjie You, Flavio Di Palo, Yiyue Qian, and Ayush Jain. 2025. Verification-guided context optimization for tool calling via hierarchical llms-as-editors. *arXiv preprint arXiv:2512.13860*.

Shih-Yang Liu, Xin Dong, Ximing Lu, Shizhe Diao, Peter Belcak, Mingjie Liu, Min-Hung Chen, Hongxu Yin, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and 1 others. 2026a. Gdpo: Group reward-decoupled normalization policy optimization for multi-reward rl optimization. *arXiv preprint arXiv:2601.05242*.

Yantao Liu, Zijun Yao, Rui Min, Yixin Cao, Lei Hou, and Juanzi Li. 2025a. Rm-bench: Benchmarking reward models of language models with subtlety and style. In *ICLR*, volume 2025, pages 44323–44355.

Yuhao Liu, Liang Zeng, Yuzhen Xiao, Jujie He, Jiacai Liu, Chaojie Wang, Rui Yan, Wei Shen, Fuxiang Zhang, Jiacheng Xu, and 1 others. 2026b. Skywork-reward-v2: Scaling preference data curation via human-ai synergy. In *ICLR*, volume 2026, pages 133805–133838.

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*.

Yining Lu, Zilong Wang, Shiyang Li, Xin Liu, Changlong Yu, Qingyu Yin, Zhan Shi, Zixuan Zhang, and Meng Jiang. 2026. Learning to optimize multi-objective alignment through dynamic reward weighting. *Transactions of the Association for Computational Linguistics*, 14:1051–1073.

Tianyi Ma, Yiyue Qian, Yiyang Li, Zehong Wang, Yifan Ding, Zheyuan Zhang, Yan Liang, Chuxu Zhang, and Yanfang Ye. 2026a. Non-monotonic autoregressive sequence model. In *ICLR*.

Tianyi Ma, Yiyue Qian, Zehong Wang, Zheyuan Zhang, Chuxu Zhang, and Yanfang Ye. 2025. Llm-empowered class imbalanced graph prompt learning for online drug trafficking detection. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 14095–14114.

Tianyi Ma, Yiyue Qian, Zheyuan Zhang, Zehong Wang, Xiaoye Qian, Feifan Bai, Yifan Ding, Xuwei Luo, Shinan Zhang, Keerthiram Murugesan, and 1 others. 2026b. Autodata: A multi-agent system for open web data collection. In *NeurIPS*, volume 38, pages 173416–173448.

Yiyue Qian, Shinan Zhang, Lanhao Chen, Diego Socolinsky, Negin Sokhandan, Song Cui, De Chen, and Suchitra Sathyanarayana. 2025. Enhancing e-commerce representation learning via hypergraph contrastive learning and interpretable llm-driven analysis. In *Companion Proceedings of the ACM on Web Conference 2025*, pages 2512–2520.

Yiyue Qian, Shinan Zhang, Huan Song, Negin Sokhandan, Hannah Marlowe, and Diego Socolinsky. 2026a. Brick-dicl: Dynamic in-context learning for automated brick schema classification. *arXiv preprint arXiv:2606.17637*.

Yiyue Qian, Shinan Zhang, Yun Zhou, Haibo Ding, Diego Socolinsky, and Yi Zhang. 2026b. Collabeval: Enhancing llm-as-a-judge via multi-agent collaboration. *arXiv preprint arXiv:2603.00993*.

Swarnadeep Saha, Omer Levy, Asli Celikyilmaz, Mohit Bansal, Jason Weston, and Xian Li. 2024. Branch-solve-merge improves large language model evaluation and generation. In *NAACL*, pages 8352–8370.

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*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Huan Song, Deeksha Razdan, Yiyue Qian, Arijit Ghosh Chowdhury, Parth Patwa, Aman Chadha, Shinan Zhang, Sharlina Keshava, and Hannah Marlowe. 2025. Learning from generalization patterns: An evaluation-driven approach to enhanced data augmentation for fine-tuning small language models. *arXiv preprint arXiv:2510.18143*.

Sijun Tan, Siyuan Zhuang, Kyle Montgomery, William Tang, Alejandro Cuadron, Chenguang Wang, Raluca Popa, and Ion Stoica. 2025. Judgebench: A benchmark for evaluating llm-based judges. In *ICLR*, volume 2025, pages 63277–63303.

Haoxiang Wang, Wei Xiong, Tengyang Xie, Han Zhao, and Tong Zhang. 2024a. Interpretable preferences via multi-objective reward modeling and mixture-of-experts. In *EMNLP*, pages 10582–10592.

Peiyi Wang, Lei Li, Liang Chen, Zefan Cai, Dawei Zhu, Binghuai Lin, Yunbo Cao, Lingpeng Kong, Qi Liu, Tianyu Liu, and 1 others. 2024b. Large language models are not fair evaluators. In *ACL*, pages 9440–9450.

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

Chenxi Whitehouse, Tianlu Wang, Ping Yu, Xian Li, Jason E Weston, Ilia Kulikov, and Swarnadeep Saha. 2026. J1: Incentivizing thinking in llm-as-a-judge via reinforcement learning. In *ICLR*, volume 2026, pages 10397–10420.

Renhao Xue, Rui Wang, Yawei Wang, Yueying Cui, Yiyue Qian, Praneetha Vaddamanu, Huan Song, and Hannah Marlowe. 2026. Inference-time model steering via predictive-state intervention: A survey.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Rui Yang, Ruomeng Ding, Yong Lin, Huan Zhang, and Tong Zhang. 2024. Regularizing hidden states enables learning generalizable reward model for llms. In *NeurIPS*, volume 37, pages 62279–62309.

Taolin Zhang, Maosong Cao, Alexander Lam, Songyang Zhang, and Kai Chen. 2025. CompassJudger-2: Towards generalist judge model via verifiable rewards. *arXiv preprint arXiv:2507.09104*.

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. In *NeurIPS*, volume 36, pages 46595–46623.

Yaowei Zheng, Junting Lu, Shenzhi Wang, Zhangchi Feng, Dongdong Kuang, Yuwen Xiong, and Richong Zhang. 2025. Easyr1: An efficient, scalable, multi-modality rl training framework.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, and Zheyan Luo. 2024. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 3: system demonstrations)*, pages 400–410.

Appendix

This appendix provides supplementary material for the main paper. Appendix A details the reward function scoring. Appendix B lists the full training framework. Appendix C presents experiment details including hyperparameters, benchmarks, and evaluation metrics. Appendix D provides qualitative case examples.

A Reward Function Details

We design $K = 4$ reward functions that jointly capture both judgment correctness and reasoning process quality. Unlike existing approaches that use a single binary outcome reward (Whitehouse et al., 2026; Zhang et al., 2025), our multi-aspect design provides fine-grained learning signals: R_{out} diagnoses where errors originate in the reasoning chain; R_{dis} ensures genuine deliberation independent of outcome; R_{eff} penalizes counterproductive discussion that corrupts correct initial judgments; and R_{fmt} enforces the structural format necessary for the other rewards to be computed. Together, these rewards create complementary training signals that cannot be captured by any single reward function.

Fine-Grained Outcome Reward R_{out}. Rather than a binary correct/incorrect signal, this reward diagnoses *where* the reasoning went wrong: whether the error originated in the initial evaluation, was introduced during discussion, or occurred at the final judgment stage. This provides partial credit when the panel was on the right track. Let $\hat{y}_i$ denote the final verdict, y_i the ground truth, $\boldsymbol{v}_{\text{init}} = \{v_i^{(1)}, \ldots, v_i^{(M)}\}$ the initial speaker verdicts, and $\boldsymbol{v}_{\text{last}}$ the verdicts from the last stage before the final result. $\text{maj}(\cdot)$ denotes the majority verdict.

$$R_{\text{out}}(\tau_{i,j}) = \begin{cases} 1.0 & \hat{y}_i = y_i, \\ 0.7 & \hat{y}_i \neq y_i,\ \text{maj}(\boldsymbol{v}_{\text{last}}) = y_i, \\ 0.5 & \hat{y}_i \neq y_i,\ \text{maj}(\boldsymbol{v}_{\text{init}}) = y_i, \\ 0.3 & \hat{y}_i \neq y_i,\ y_i \in \boldsymbol{v}_{\text{init}}, \\ 0.0 & \text{otherwise.} \end{cases} \tag{5}$$

Discussion Quality Reward R_{dis}. This reward measures whether the multi-round discussion produces genuine deliberation that contributes to correct outcomes. Let d denote the number of discussion rounds and Δ_{score} indicate whether any speaker changed their verdict:

$$r_{\text{disc}} = \begin{cases} 1.0 & d = 0,\ \text{unanimous},\ \hat{y} = y, \\ 0.0 & d = 0,\ \text{unanimous},\ \hat{y} \neq y, \\ 0.2 & d = 0,\ \neg\,\text{unanimous (initial split)}, \\ 0.2 & d > 0,\ \mathbf{v}^{(d)}\ \text{unparseable}, \\ 1.0 & d > 0,\ \Delta_{\text{score}},\ \hat{y} = y, \\ 0.3 & d > 0,\ \Delta_{\text{score}},\ \hat{y} \neq y, \\ 0.3 & d > 0,\ \neg\,\Delta_{\text{score}}. \end{cases} \tag{6}$$

Intuitively, the two extreme signals are the unanimous no-discussion cases: a panel that already agrees and is right (1.0) needs no deliberation, while a confidently-wrong consensus (0.0) is maximally penalized. When the panel is split but skips discussion ($d = 0$, not unanimous), it receives a low 0.2 for failing to deliberate. Once discussion occurs ($d > 0$), the panel earns the full 1.0 only when scores actually change and the final verdict is correct, rewarding productive revision; a discussion that changes scores but stays wrong, or that merely restates positions without any change, both receive 0.3. The residual 0.2 covers rounds whose scores cannot be parsed.

Efficiency Reward R_{eff}. This reward evaluates whether the model's decision to engage in or skip discussion was *beneficial* to the final outcome. The strongest penalty (-0.3) targets the most harmful failure: when initial speakers had the correct majority but discussion corrupted it, directly discouraging counterproductive deliberation.

$$R_{\text{eff}}(\tau_{i,j}) = \begin{cases} 1.0 & \hat{y}_i = y_i,\ \text{unanimous},\ d{=}0, \\ 0.8 & \hat{y}_i = y_i,\ \text{discussion corrected}, \\ 0.3 & \hat{y}_i = y_i,\ \text{otherwise}, \\ -0.3 & \hat{y}_i \neq y_i,\ \text{flipped correct}, \\ 0.0 & \hat{y}_i \neq y_i,\ \text{otherwise}. \end{cases} \tag{7}$$

Format Compliance Reward R_{fmt}. Computed as a weighted sum of binary checks for: (i) required section tags (e.g., `[Initial Evaluation]`, `[Final Evaluation Result]`), (ii) presence of M speakers with complete evaluation fields (strengths/weaknesses, per-criterion scores, justification, confidence), and (iii) proper discussion and final judge structure when present.

B Algorithm

Algorithm 1 summarizes the complete JudgePanel training pipeline, including panel data construc-

tion, SFT, AdaReward reinforcement learning with adaptive weight updates, and lightweight domain specialization.

Algorithm 1 JudgePanel: Training Pipeline

Input: Base model π_{θ_0}; panel model $\mathcal{P}$; arena data $\mathcal{D}_{\text{arena}}$; rewards $\{R_k\}_{k=1}^{K}$; targets $\{\rho_k\}$
Output: Domain-specialized judge π_θ

```
Stage 1: Panel Data Construction (§3.1)
Run panel protocol on D_arena; retain correct traces → D_panel
Stage 2: SFT
Fine-tune π_θ on D_panel
Stage 3: AdaReward RL (§3.2)
for step s = 1, ..., S do
    Sample rollouts, compute rewards {R_k(τ_{i,j})}
    Compute Â_sum^{(i,j,s)} via adaptive weights
    Update π_θ via J_Ada (Eq. 4)
    if s mod Δ = 0 then
        Update weights viaEq. 2–3
    end if
end for
Stage 4: Domain Specialization (§3.3)
LoRA fine-tune on domain-specific panel traces (200–500 samples)
return π_θ
```

C Experiment Details

This section provides implementation details for reproducing our results, including training hyper-parameters for all three stages, benchmark descriptions, domain specialization data sources, and evaluation metric definitions.

C.1 Hyper-Parameters

For SFT, we use LlamaFactory [1] (Zheng et al., 2024) train for 3 epochs with learning rate 1×10^{-6}, patience 10, cosine schedule, batch size 32, and max sequence length 8,192. For AdaReward RL, we train for 5 epochs with G=8 rollouts per prompt, learning rate 5×10^{-7}, clip ratio ε=0.2, and KL penalty 0.08. The adaptive learner uses: initial weights $w_{\text{out}}^{(0)}$=5.0, $w_{\text{disc}}^{(0)}$=2.0, $w_{\text{fmt}}^{(0)}$=2.0, $w_{\text{eff}}^{(0)}$=2.0, targets ρ_{out}=0.95, ρ_{dis}=0.85, ρ_{eff}=0.85, ρ_{fmt}=0.99, scaling α^{+}=0.5, α^{-}=0.1, momentum β=0.3, bounds $[3, 10]$ for outcome and $[1, 4]$ for other rewards, and update interval Δ=5. For DA, we apply LoRA (r=16, α=32) with learning rate 1×10^{-4} and early stop with patient 10 for 3 epochs on domain-specific samples. All experiments are conducted on 8× NVIDIA H200 (143GB) GPUs using EasyR1[2] (Zheng et al., 2025) for the implementation of GDPO and GRPO. SFT training takes approximately 2 hours, RL training takes approximately 22 hours, and LoRA DA takes approximately 5 minutes per task.

C.2 Benchmark and DA Data Details

Benchmarks. (1) JudgeBench (Tan et al., 2025): 620 challenging response pairs spanning knowledge, reasoning, math, and coding, evaluated by objective correctness. (2) RewardBench (Lambert et al., 2025): 2,985 prompt-chosen-rejected trios covering chat, reasoning, and safety. (3) RM-Bench (Liu et al., 2025a): 3,981 samples testing reward model sensitivity to subtle content differences and resistance to style biases. (4) PPE (Frick et al., 2025): 16,038 samples evaluating pairwise preference alignment across 12 domains with multilingual coverage. For fair comparison, all models use *parsed accuracy*, computed as correct predictions divided by total samples excluding unparseable outputs.

C.3 Evaluation Metric Definitions

Avg. Rank (Table 1): For each benchmark, models are ranked by accuracy (rank 1 = highest). Avg. Rank is the mean rank across all four benchmarks (lower is better).
Consistency (Table 2): The percentage of samples where the model's verdict correctly tracks position swaps, i.e., if the model prefers Response A in the original order, it should prefer Response B when the order is swapped.
Stable Accuracy (Table 2): The percentage of samples correctly predicted in *both* orderings, measuring position-invariant judgment quality.
Constructive Change Rate (Table 3a): Among samples where at least one speaker changes their vote during discussion, the percentage that end up with the correct final answer. Formally: $|\{\text{vote changed} \cap \text{final correct}\}|/|\{\text{vote changed}\}|$. A rate near 50% indicates random changes; significantly above 50% indicates deliberation is functional.
Jaccard Distance (Table 3a, speaker-level, ↑): Computed as $1 - |T_i \cap T_j|/|T_i \cup T_j|$ over character

[1] http://github.com/hiyouga/LlamaFactory
[2] https://github.com/hiyouga/EasyR1

3-grams between each pair of speakers within the same round, then averaged across all pairs. Higher values indicate speakers use more distinct phrasing and reasoning paths.

4-gram Overlap (Table 3a, phase-level, ↓): The fraction of word 4-grams in discussion rounds that also appear in the initial round: $|N_{\text{disc}} \cap N_{\text{init}}|/|N_{\text{disc}}|$. Lower values indicate discussion introduces novel content rather than repeating prior reasoning.

Average Rounds (Table 3a): The mean number of discussion rounds generated per sample. The model should adaptively engage in discussion only when speakers initially disagree.

D Case Examples

We provide qualitative examples comparing panel trace generation with single-model COT generation on the same evaluation task, illustrating how multi-agent deliberation enables disagreement resolution that single-model reasoning cannot achieve.

D.1 Single COT Think Trace

Below is a single-model COT thinking trace generated by Qwen model on the same coding evaluation task, following the format of independent evaluation of a single speaker. Without multi-agent discussion, the model arrives at an incorrect verdict.

Input

Instruction: You are given two integers num1 and num2. In one operation, you can choose integer i in the range $[0, 60]$ and subtract $2^i + \texttt{num2}$ from num1. Return the minimum number of operations needed to make num1 equal to 0. If impossible, return -1.

Response A:

```
for k in range(61):
  target = num1 - k * num2
 if target >= 0 and bin(target).count('1') <= k:
    return k
return -1
```

Response B:

```
for k in range(61):
  target = num1 - k * num2
  if target < 0: continue
 if bin(target).count('1') <= k and k <= target:
    return k
return -1
```

Single COT Evaluation

Let me analyze both solutions. After k operations, we need $\texttt{num1} - k \cdot \texttt{num2} = \Sigma(2^{i_j})$. The minimum powers of two needed equals the popcount. Both solutions check this condition. Response B adds `k <= target` and explicit negative-target skip. However, `k <= target` appears redundant within problem constraints ($\texttt{num1} \leq 10^9$): when num2 is negative, target grows with k; when positive, the loop terminates early. Response A captures the essential logic without unnecessary conditions.

Strengths and Weaknesses of Response A: Elegant and minimal. Correctly identifies the two core conditions (non-negative target and bit-count bound). No unnecessary imports or redundant checks.

Strengths and Weaknesses of Response B: Functionally correct but over-engineered. The `k <= target` condition is redundant within problem constraints, and the unused `math` import indicates incomplete code cleanup.

Score of Response A: Total: 42. Accuracy: 9, Relevance: 9, Completeness: 8, Clarity: 9, Usefulness: 7.

Score of Response B: Total: 36. Accuracy: 8, Relevance: 9, Completeness: 7, Clarity: 7, Usefulness: 5.

Potential Errors: Edge cases where num2 is positive and large could make target small relative to k, but the `target >= 0` guard handles termination.

Evaluation Result: model_a

Justification: Response A is preferred for its clean, minimal implementation that captures exactly the necessary and sufficient conditions.

D.2 Panel Trace Example

Below is a full panel reasoning trace generated by our multi-agent protocol on the same coding evaluation task. In contrast to the single COT above, the panel protocol enables disagreement resolution: Speaker 2 initially favors model_a (similar to the single COT reasoning), but after hearing Speaker 1's argument about Python's `bin()` behavior on negatives during discussion, revises to model_b. All three speakers then reach consensus with maximum confidence—demonstrating how multi-agent deliberation corrects individual reasoning errors.

Input

Instruction: You are given two integers num1 and num2. In one operation, you can choose integer i in the range $[0, 60]$ and subtract $2^i + \texttt{num2}$ from num1. Return the minimum number of operations needed to make num1 equal to 0. If impossible, return -1.

Response A:

```
for k in range(61):
  target = num1 - k * num2
 if target >= 0 and bin(target).count('1') <= k:
    return k
return -1
```

Response B:

```
for k in range(61):
  target = num1 - k * num2
  if target < 0: continue
 if bin(target).count('1') <= k and k <= target:
    return k
return -1
```

Phase 1: Independent Evaluation

[Speaker 1] Score: model_b | Confidence: 5
After k operations, $\Sigma(2^{i_j}) = \texttt{num1} - k \cdot \texttt{num2}$. Each term $2^{i_j} \geq 1$, so target $\geq k$. Response A has a bug: Python's `bin()` on negatives returns `'-0b...'` where `count('1')` yields 0, falsely satisfying the condition. Response B explicitly skips negatives and adds safe $k \leq$ target bound.
Strengths and Weaknesses of Response A: Concise and clean but lacks proper handling of negative targets—Python's `bin()` on negative numbers produces unexpected output, leading to incorrect acceptance of invalid cases.
Strengths and Weaknesses of Response B: More robust by explicitly skipping negative targets. The added `k <= target` condition is technically redundant but harmless.
Score of Response A: Total: 33. Accuracy: 6, Relevance: 8, Completeness: 7, Clarity: 8, Usefulness: 4.
Score of Response B: Total: 40. Accuracy: 9, Relevance: 8, Completeness: 8, Clarity: 8, Usefulness: 7.
Potential Errors: One might overlook the subtle issue with Python's `bin()` behavior on negative numbers.
Evaluation Result: model_b
Justification: Response B avoids the Python-specific bug where `bin()` on negative numbers produces malformed strings that incorrectly satisfy the bit-count condition.

[Speaker 2] Score: model_a | Confidence: 5
Both solutions use the same core algorithm. Response A is cleaner and more concise. The `k <= target` condition in Response B appears redundant—since `bin(target).count('1') <= k` already implies the target has enough bits to decompose into k powers. Response B also has an unnecessary `import math`. The extra condition adds complexity without clear benefit.
Strengths and Weaknesses of Response A: Clean, minimal implementation. Directly checks the two necessary conditions without unnecessary additions. Easy to read and maintain.
Strengths and Weaknesses of Response B: Adds a seemingly redundant `k <= target` check and an unused import. While not incorrect, the extra complexity is unjustified and suggests the author may not fully understand why the condition is unnecessary.
Score of Response A: Total: 40. Accuracy: 8, Relevance: 9, Completeness: 8, Clarity: 9, Usefulness: 6.
Score of Response B: Total: 36. Accuracy: 7, Relevance: 9, Completeness: 7, Clarity: 7, Usefulness: 6.
Potential Errors: May be underestimating edge cases where the redundant condition actually matters.
Evaluation Result: model_a
Justification: Response A is preferred for its simplicity and directness. The additional checks in B appear unnecessary and add complexity without improving correctness.

[Speaker 3] Score: model_b | Confidence: 4
Response A lacks an explicit check for the case where k exceeds the target, which could lead to incorrect results. Response B improves upon A by including this additional check.
Strengths and Weaknesses of Response A: Straightforward approach but lacks explicit check for $k >$ target scenario.
Strengths and Weaknesses of Response B: More comprehensive and accurate by including the additional $k \leq$ target check.
Score of Response A: Total: 36. Accuracy: 8, Relevance: 9, Completeness: 6, Clarity: 8, Usefulness: 5.
Score of Response B: Total: 42. Accuracy: 9, Relevance: 9, Completeness: 8, Clarity: 8, Usefulness: 8.
Potential Errors: Not considering the scenario where k exceeds the target could lead to incorrect conclusions.
Evaluation Result: model_b
Justification: Response B is better due to its improved completeness and accuracy from the additional check preventing $k >$ target errors.

→ No consensus (S1, S3: model_b; S2: model_a). Proceeding to discussion.

Phase 2: Discussion (Round 1)

[Speaker 1] Revised Score: model_b | Confidence: 5
Disagreement Resolution: Speaker 2 argues that `k <= target` is redundant and prefers A for conciseness. However, this overlooks the critical Python-specific bug: `bin()` on negative numbers returns `'-0b101'` whose `.count('1')` yields 0, falsely satisfying the condition. Response A's `target >= 0` check does guard against this, but combined with the missing $k \leq$ target bound, A has two potential failure modes.
Revised Score of Response A: Total: 34. Accuracy: 6, Relevance: 8, Completeness: 7, Clarity: 8, Usefulness: 5.
Revised Score of Response B: Total: 41. Accuracy: 9, Relevance: 9, Completeness: 9, Clarity: 8, Usefulness: 8.
Justification: Response B's explicit handling of negative targets and adherence to all logical requirements outweigh conciseness concerns raised by Speaker 2.

[Speaker 2] Revised Score: model_b | Confidence: 5
Disagreement Resolution: I originally overweighted conciseness and claimed `k <= target` was redundant. Speaker 1 correctly identified a deeper issue I missed: Python's `bin()` on negative numbers returns strings like `-0b101` whose `.count('1')` yields 0, falsely satisfying the condition. While Response A does check `target >= 0`, the combination of this subtle behavior with the missing $k \leq$ target bound creates real correctness vulnerabilities. I am revising my position from model_a to model_b.
Revised Score of Response A: Total: 33. Accuracy: 5, Relevance: 8, Completeness: 7, Clarity: 8, Usefulness: 5.
Revised Score of Response B: Total: 43. Accuracy: 9, Relevance: 9, Completeness: 9, Clarity: 8, Usefulness: 8.
Justification: Speakers 1 and 3 correctly identified that Response B covers all bases safely. Response A's absence of an explicit `target < 0` guard and missing $k \leq$ target creates correctness vulnerabilities that outweigh its conciseness advantage.

[Speaker 3] Revised Score: model_b | Confidence: 5
Disagreement Resolution: Speaker 2 initially preferred Response A for conciseness but has now revised after acknowledging the `bin()` pitfall identified by Speaker 1. I concur—the omitted negative-target guard in A is a concrete bug. Raising confidence from 4 to 5 given full consensus.
Revised Score of Response A: Total: 33. Accuracy: 5, Relevance: 9, Completeness: 6, Clarity: 8, Usefulness: 5.
Revised Score of Response B: Total: 44. Accuracy: 9, Relevance: 9, Completeness: 9, Clarity: 8, Usefulness: 9.
Justification: All speakers recognized that Response B handles both the Python-specific pitfall and the required `k <= target` bound, making it the only solution guaranteed to avoid false positives.

→ *Consensus reached. All speakers agree on model_b with confidence 5. Final evaluation result:*

Final Evaluation Result

Score: model_b | Confidence: 5