JudgeRLVR: Judge First, Generate Second for Efficient Reasoning

Anonymous ACL submission

Abstract

Reinforcement Learning with Verifiable Rewards (RLVR) has become a standard paradigm for reasoning in Large Language Models. However, optimizing solely for final-answer correctness often drives models into aimless, verbose exploration, where they rely on exhaustive trial-and-error tactics rather than structured planning to reach solutions. While heuristic constraints like length penalties can reduce verbosity, they often truncate essential reasoning steps, creating a difficult trade-off between efficiency and verification. In this paper, we argue that discriminative capability is a prerequisite for efficient generation: by learning to distinguish valid solutions, a model can internalize a guidance signal that prunes the search space. We propose JudgeRLVR, a two-stage *judge-then-generate* paradigm. In the first stage, we train the model to judge solution responses with verifiable answers. In the second stage, we fine-tune the *same* model with vanilla generating RLVR initialized from the judge. Compared to Vanilla RLVR using the same math-domain training data, JudgeRLVR achieves a better quality–efficiency trade-off on both MoE (Qwen3-30B-A3B) and dense (Qwen3-8B) architectures: on **in-domain math**, it delivers up to **+3.7 points** average accuracy gain with up to **42%** mean per-benchmark length reduction; on **out-of-domain** benchmarks, it delivers up to **+4.5 points** average accuracy improvement.

1 Introduction

Reinforcement Learning with Verifiable Rewards (RLVR) has become a widely adopted method for improving large language models on complex reasoning tasks (Zhu et al., 2025; Shao et al., 2025). By providing sparse yet automatically checkable supervision signals (e.g., whether the final answer is correct), RLVR enables models to explore strategies beyond supervised demonstrations and has achieved strong results on reasoning benchmarks like math and code.

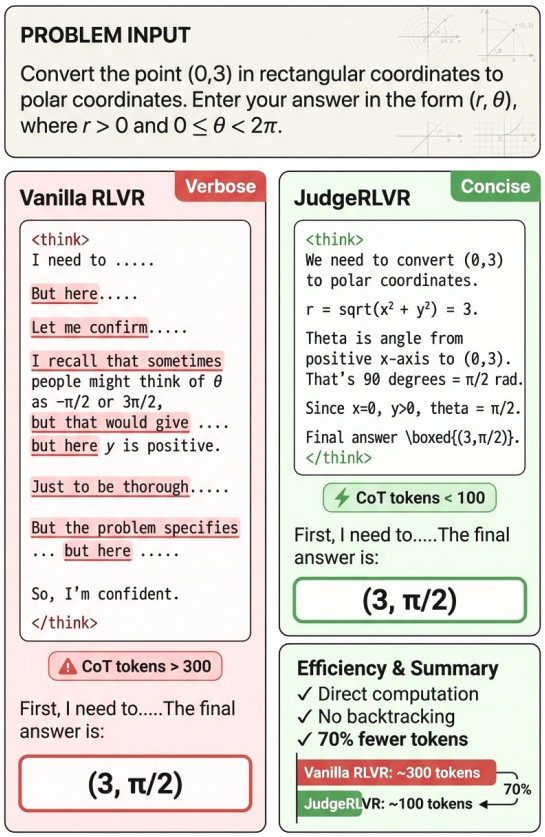

Figure 1: Qualitative comparison of reasoning patterns between Vanilla RLVR and JudgeRLVR. While outcome-based RL guarantees final answer correctness, it often induces verbose and unstructured exploration (left). JudgeRLVR implicitly regularizes the reasoning process, leading to a direct and coherent solution path (right).

However, RLVR does not merely optimize correctness; it also implicitly shapes a model's *thinking pattern*. When trained primarily with a final-answer reward, many LLMs drift toward a generative search style: enumerating tentative branches, revising intermediate steps, and performing explicit self-correction (Team et al., 2024). Figure 1 shows that the reasoning model's output can be divided

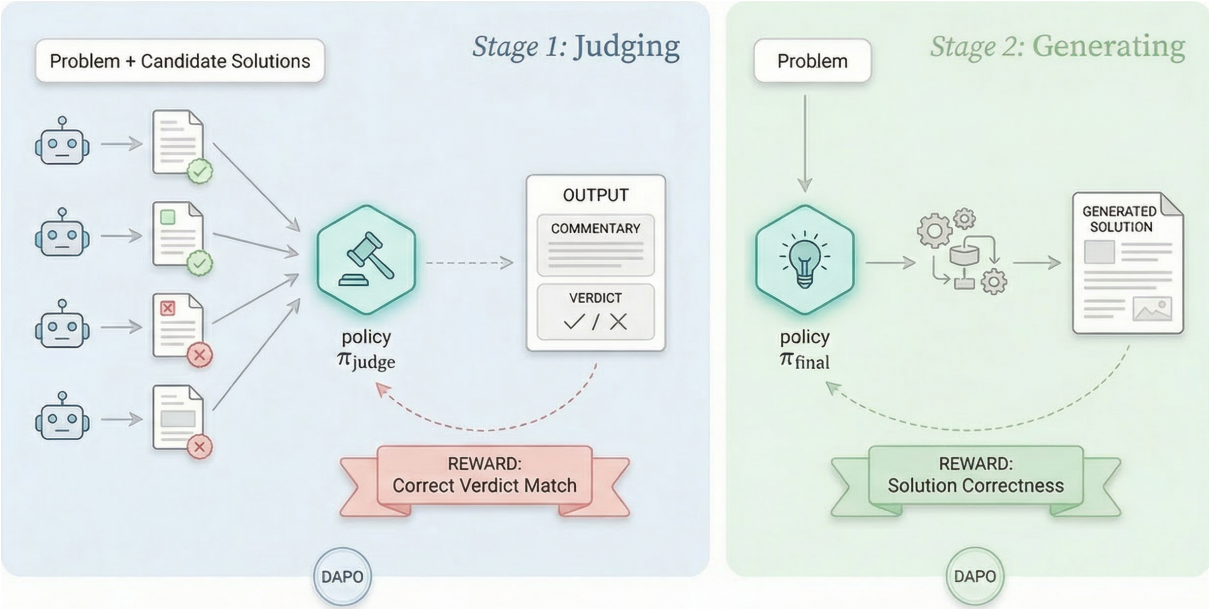

Figure 2: Pipeline for the two-stage training of JudgeRLVR

into 2 parts: chain-of-thought (CoT) trajectories and solution responses. CoT trajectories usually include all the details of the model's lengthy thought process, whereas solution responses tend to present a clear logical process along with the final answer.

The resulting CoT trajectories are often long and exhibit frequent backtracking (e.g., heavy use of "but", "however", "let's try again") (Cai and et al., 2025). Long outputs are expensive, but a deeper issue is *quality*: such trajectories are low in information density. Existing methods such as Kimi K1.5 (Moonshot AI Team, 2025) and DAPO (Yu et al., 2025) introduce length penalties to reduce tokens and stabilize training, but often come with accuracy drops, making it difficult to achieve an ideal quality–efficiency trade-off.

Cognitive science suggests that expert reasoning relies on *early discrimination and pruning*—filtering unpromising paths before expanding them—rather than exhaustive search (Chi et al., 1981). This motivates the idea that, to achieve efficient reasoning, one should first learn to judge what constitutes valid reasoning, and then learn how to generate solutions.

Based on the idea above, we propose **JudgeRLVR**, a two-stage *judge-then-generate* paradigm. As shown in Figure 2, in the judging stage, the model is trained to evaluate candidate solution responses generated by diverse models and predict whether they are correct or incorrect. This stage is intended to build error sensitivity and a preference for direct, valid solution patterns. In the generating stage, we initialize from the judge and apply *Vanilla RLVR* fine-tuning on the same model, so that generation is optimized from a starting point that is already better at distinguishing productive from unproductive reasoning paths.

Crucially, we impose no explicit length or quality penalties on CoT trajectories; thus, any reduction in verbosity is interpreted as arising from the discriminative prior acquired in the judging stage, rather than from direct length control.

Compared to Vanilla RLVR using the same training data, JudgeRLVR achieves a quality–efficiency trade-off on both Qwen3-30B-A3B (MoE) (Team, 2025) and Qwen3-8B (dense) (Team, 2025): on **in-domain math**, it delivers up to **+3.7 points** average accuracy improvement with up to **42%** mean per-benchmark length reduction; on **out-of-domain** benchmarks, it delivers up to **+4.5 points** average accuracy improvement.

In summary, our main contributions are: (i) we propose **JudgeRLVR**, a new training paradigm for inducing higher-quality thinking patterns via *judge-then-generate*; (ii) JudgeRLVR demonstrates a better quality–efficiency trade-off than Vanilla RLVR on both in-domain and out-of-domain tasks; (iii) we validate the paradigm on both MoE and dense architectures, and provide ablation evidence that the discriminative RL objective—rather than mere warm-start—is a key driver of improvement; and (iv) we provide interpretable linguistic evidence consistent with reduced explicit backtracking.

2 Related Work

2.1 RLVR and Verifiable-Reward Training for Reasoning

Reinforcement Learning with Verifiable Rewards (RLVR) is a leading approach for improving LLM reasoning using automatic, task-specific rewards, often outperforming supervised fine-tuning on math and logic tasks (Lambert et al., 2024; Guo et al., 2025a; Xu et al., 2025). However, RLVR has notable limitations: it can yield gains even under spurious rewards (Shao et al., 2025), and it mainly reweights existing high-reward reasoning paths rather than inducing new reasoning behaviors (Yue et al., 2025). This is consistent with concerns that RLVR may implicitly encourage suboptimal thinking by optimizing final correctness instead of process quality (DataLearner, 2025). Most variants emphasize reward design or optimization, leaving the quality–efficiency trade-off underexplored.

2.2 Mathematical Reasoning in Large Language Models

Mathematical reasoning is a core benchmark for LLMs, where recent progress relies heavily on RLVR and targeted training (Team et al., 2024; Shen et al., 2025). Data diversification via paraphrasing can improve generalization (Shen et al., 2025), while extended RLVR training improves accuracy but tends to produce verbose reasoning traces (Team et al., 2024). Synthetic data scaling further boosts capability yet does not eliminate redundancy (Zhou et al., 2025), and scaling studies still report low information density in reasoning outputs (Sun et al., 2024). Overall, improving both correctness and efficiency remains challenging.

2.3 Learning to Judge: Evaluators, Verifiers, and Discriminative Supervision

Training evaluators and using discriminative supervision can steer LLM outputs by separating good from bad generations (Guo et al., 2025b; Tang et al., 2025). Discriminative fine-tuning can suppress negative outputs without preference data (Guo et al., 2025b), and stronger evaluators improve reasoning-quality assessment via coherence-focused criteria (Tang et al., 2025). Verifiers can detect reasoning errors with uncertainty signals (Han et al., 2024), and systematic studies highlight the need for task-specific validation metrics (Shimabucoro et al., 2024). Yet existing work rarely adopts a two-stage pipeline that first trains judgment and then uses it to guide generation, which we address in JudgeRLVR.

3 Method

JudgeRLVR divides the training of a single reasoning policy into two sequential stages with different roles but intended capability transfer: a *judge* stage that trains discriminative error awareness, followed by a *generating* stage that optimizes solution generation. Our core hypothesis is that discriminative competence is a prerequisite for efficient generation. Once the model internalizes what constitutes a valid CoT trajectory, it can down-weight erroneous branches early in generation without any explicit length penalty.

Figure 2 illustrates the complete pipeline of our two-stage training.

3.1 Notation and Task Definition

The dataset consists of problems in a specific domain: $x \in \mathcal{X}$ with gold final answers $a^\star(x)$.

The **solution response** is denoted $z = (o_1, \ldots, o_T)$, a token sequence containing a clear logical process and ending with a final answer string. The final answer $a(z)$ is extracted from the solution responses via a deterministic parser. We do not train the model to judge directly on full outputs that include complete CoT trajectories because such outputs are often extremely long and contain a large amount of distracting information that can interfere with the judgment. Instead, we expect the model to judge directly based on the clean solution response, so that it can identify errors with greater precision.

We define a binary correctness label $\ell \in \{0, 1\}$:

$$\ell(x, z) = \mathbb{I}\big(a(z) = a^\star(x)\big). \tag{1}$$

The policy (language model) is denoted π_θ.

3.2 Stage 1: Judging Stage

In judging stage, the model is trained as a judge. Given a problem x and a candidate solution response z, the policy produces a critique/commentary c which contains the CoT trajectory, and outputs a discrete verdict token $v \in \{0, 1\}$ (0 for incorrect and 1 for correct):

$$(c, v) \sim \pi_\theta(\cdot \mid x, z). \tag{2}$$

Data construction. We construct a discriminative dataset D_{judge} consisting of triplets (x, z, ℓ):

- **Response generation (rollout).** For each problem x, we sample a set of candidate solution responses $\{z_1, \ldots, z_n\}$ from several models, and obtain $\ell(x, z_i)$ by comparing $a(z_i)$ with $a^\star(x)$.
- **Hard negative mining.** We prioritize *moderate-difficulty* problems whose empirical pass rates under rollouts are neither 0 nor 1, yielding more informative "nearly-correct" errors.
- **Class balancing.** For each x, we subsample to balance positives ($\ell = 1$) and negatives ($\ell = 0$), avoiding majority-class bias.

Reward. The reward is whether the final verdict v_i matches the true label $\ell(x, z_i)$:

$$r_i = \mathbb{I}(v_i = \ell(x, z_i)) \tag{3}$$

The training objective encourages v_i to match the true label $\ell(x, z_i)$. Meanwhile, the commentary c, as an explanatory prefix, is optimized by the same policy-gradient signal. Note that both stages employ the same RL training algorithm (DAPO); the key distinction is the task formulation—judging in Stage 1 versus generating in Stage 2—not the optimization paradigm.

3.3 Stage 2: Generating Stage

In generating stage, the model is trained under Vanilla RLVR setting and outputs solutions of the problems. The policy is initialized from the judging stage weights.

Given a problem x, the policy generates a CoT trajectory and the solution response z (including the final parsed answer $a(z)$):

$$z \sim \pi_\theta(\cdot \mid x). \tag{4}$$

The reward remains the sparse binary correctness signal based only on the final answer:

$$r = \ell(x, z) = \mathbb{I}\big(a(z) = a^\star(x)\big). \tag{5}$$

3.4 Hypothesized Mechanism and Verification Overview

We hypothesize that JudgeRLVR improves reasoning-pattern quality through the two stages in two corresponding ways: (i) **style transfer**: in judging stage, learning to judge induces a global shift in the model's reasoning language style; (ii) **reduced backtracking**: in generating stage, training activates the internalized efficient pattern, reducing explicit self-correction and backtracking in text. We design targeted experiments to test both points; definitions and results are provided in Section 4 and 5.

4 Experimental Setup

4.1 Model Configuration and Training Data

We experiment with two model architectures to evaluate JudgeRLVR under different settings. (1) **Qwen3-30B-A3B**, a mixture-of-experts (MoE) architecture with 30B total and 3B active parameters; we conduct supervised fine-tuning (SFT) on curated open-source CoT datasets to obtain Qwen3-30B-A3B-SFT as the Base SFT checkpoint. (2) **DeepSeek-R1-0528-Qwen3-8B** (DeepSeek-AI, 2025), a dense 8B-parameter model distilled from DeepSeek-R1-0528 onto the Qwen3-8B architecture; we directly adopt it as the Base SFT checkpoint. The two models differ in architecture (MoE vs. dense), scale (3B active vs. 8B), and initialization source (self-SFT vs. distilled). Both use identical training data, hyperparameters, and total training steps for Vanilla RLVR and JudgeRLVR.

For RLVR, we collect and filter open-source resources to build a dataset of 113k math problems with gold answers. To construct specialized training data for the judging stage, we adopt a strict sampling strategy:

- **Rollout generation.** For each problem, we use MiMo-7B RL (Xiaomi, 2025) and Qwen3-30B-A3B-SFT to each generate 8 reasoning traces and answers, yielding 16 distinct reasoning paths.
- **Balanced sampling.** We filter outputs so that each problem retains an equal number of correct and incorrect solutions, avoiding bias from class imbalance.

Our training algorithm is DAPO (Yu et al., 2025), a GRPO-family policy gradient method. We implement dynamic sampling to filter out samples with pass rates 0 or 1 during rollout. The batch size is set to 256 and the rollout size is $n = 8$. We use AdamW optimizer (Loshchilov and Hutter, 2018) for training. The learning rate is set to 3×10^{-6} with a 10-step warmup. We use a training temperature of 1.0 and a testing temperature

Table 1: Main results on two architectures. Acc = accuracy (%); Len = avg generation length (tokens). Δ: JudgeRLVR vs Vanilla RLVR. Overall Δ% is the mean of per-benchmark relative length changes.

Benchmark	Base SFT		Vanilla RLVR		JudgeRLVR (Seq)			
	Acc↑	Len↓	Acc↑	Len↓	Acc↑	Δ	Len↓	Δ%
Qwen3-30B-A3B (MoE, 30B total / 3B active)								
In-Domain (Math)								
AIME24	85.7	20.8k	86.3	21.8k	**89.0**	+2.7	**12.9k**	-41
AIME25	74.1	26.8k	77.0	27.2k	**78.7**	+1.7	**17.2k**	-37
MATH500	97.4	6.7k	**98.0**	6.8k	97.2	-0.8	**2.0k**	-71
HMMT_feb_2025	57.1	24.9k	60.8	26.9k	**70.0**	+9.2	**20.4k**	-24
BeyondAIME	55.5	33.8k	58.3	34.3k	**63.9**	+5.6	**21.3k**	-38
Overall Avg	74.0	22.6k	76.1	23.4k	**79.8**	+3.7	**14.8k**	-42
Out-of-Domain								
GPQA Diamond	63.0	12.0k	61.2	12.1k	**66.4**	+5.2	**11.2k**	-7.5
IFEval	80.6	**3.2k**	79.9	4.1k	**86.4**	+6.5	4.6k	+12
LiveCodeBenchv6	57.4	20.6k	58.2	22.8k	**63.9**	+5.7	**18.7k**	-18
MMLU Redux	86.8	2.1k	85.4	2.1k	**87.0**	+1.6	**1.5k**	-28
ZebraLogic	84.6	**7.7k**	83.6	7.9k	**86.8**	+3.2	9.6k	+22
Overall Avg	74.5	9.1k	73.7	9.8k	**78.1**	+4.5	**9.1k**	-3.9
Qwen3-8B (Dense, 8B)								
In-Domain (Math)								
AIME24	72.3	20.9k	73.0	21.5k	**75.3**	+2.3	**15.0k**	-30
AIME25	59.7	23.2k	62.0	24.0k	**64.0**	+2.0	**17.0k**	-29
MATH500	97.0	7.5k	**97.4**	7.6k	96.8	-0.6	**2.5k**	-67
HMMT_feb_2025	46.3	25.6k	49.2	27.0k	**56.5**	+7.3	**21.0k**	-22
BeyondAIME	32.9	28.4k	35.3	30.5k	**39.0**	+3.7	**23.0k**	-25
Overall Avg	61.6	21.1k	63.4	22.1k	**66.3**	+2.9	**15.7k**	-29
Out-of-Domain								
GPQA Diamond	62.2	8.7k	60.6	9.2k	**64.2**	+3.6	**8.5k**	-8
IFEval	73.6	**1.0k**	73.2	1.2k	**78.5**	+5.3	1.4k	+17
LiveCodeBenchv6	50.0	18.4k	50.6	19.5k	**54.8**	+4.2	**16.8k**	-14
MMLU Redux	83.7	1.9k	82.8	2.0k	**84.2**	+1.4	**1.5k**	-25
ZebraLogic	88.7	**5.8k**	87.8	6.2k	**89.8**	+2.0	7.0k	+13
Overall Avg	71.6	7.2k	71.0	7.6k	**74.3**	+3.3	**7.0k**	-8

of 0.6, with top-p set to 1.0 and max tokens set to 65536 (Qwen3-30B-A3B) and 32768 (Qwen3-8B) for both training and testing.

4.2 Evaluation Benchmarks

We evaluate JudgeRLVR on in-domain math reasoning: AIME24 (Zhang and Math-AI, 2024), AIME25 (Zhang and Math-AI, 2025), MATH500 (Hendrycks et al., 2021), HMMT_feb_2025 (Balunović et al., 2025), BeyondAIME ([ByteDance-Seed], 2025) and on diverse out-of-domain benchmarks: GPQA Diamond (Science Reasoning) (Rein et al., 2023), IFEval (Instruction Following) (Zhou et al., 2023), LiveCodeBenchv6 (Coding) (Jain et al., 2024), MMLU-Redux (General Knowledge) (Gema et al., 2025), ZebraLogic (Logical Reasoning) (Lin et al., 2025), covering a wide range of general capabilities. The detailed setups of these benchmarks are in appendix C.

4.3 Main Experiments

Our main experiments are designed to answer a single overarching question: does the proposed judge-then-generate paradigm improve the quality–efficiency trade-off of RLVR beyond Vanilla RLVR? To make this comparison meaningful, we evaluate three training settings that isolate the effect of the staged paradigm from other factors.

Base SFT: the static baseline model Qwen3-30B-A3B-SFT and DeepSeek-R1-0528-Qwen3-8B. Base SFT serves as a reference point for the model's reasoning ability before RL and its default generation style, allowing us to measure how much of the downstream gains come specifically from RLVR.

Vanilla RLVR: Vanilla generating RLVR optimizing only for final-answer correctness. Vanilla RLVR serves as a strong baseline to demonstrate the advantages of our method. The total training is 250 steps.

JudgeRLVR: our judge-then-generate method. The judging stage is 145 steps and the generating stage is 105 steps.

Vanilla RLVR and JudgeRLVR use the same training and evaluation hyperparameters and the same total training steps. We report both accuracy and generation length because our goal is not only to increase correctness, but also to reduce unnecessary computation and improve information density

in the produced reasoning.

The generating and rollout prompt, and the judging prompt are in appendix A.

4.4 Ablations

We conduct targeted ablations to validate that JudgeRLVR's gains come from the divided stages itself rather than from simply adding more training signals.

Judge Only: only judging stage training. Judge Only setting evaluates whether judge training alone can already improve generation quality, or whether generating stage is necessary to translate judging competence into improved generation performance. This ablation directly tests our core assumption that judging is a prerequisite but not a substitute for generating optimization.

Mixed Strategy: judging stage is replaced by parallel training on mixed Judge/Generative data (ratio 1:1), with each data type using its corresponding reward; generating stage is then run as usual. Mixed Strategy setting probes whether the ordering and separation of roles matters: by interleaving judge-style and generation-style updates, we test whether simultaneously optimizing two behaviors interferes with learning a clean internal decision policy. In other words, this ablation examines whether the model benefits from first consolidating an error-aware judge prior before being asked to explore and exploit it during generation.

SFT-then-RLVR: to isolate whether JudgeRLVR's gains come from the *discriminative* RL objective in Stage 1 or merely from a better warm-start via exposure to solution-response data, we replace Stage 1 with standard generative SFT. Because standard generative SFT naturally trains on positive demonstrations, SFT-then-RLVR uses the positive subset of the Stage-1 judge dataset as a warm-start control. Stage 2 proceeds identically with Vanilla RLVR. This comparison isolates the role of the Stage 1 training paradigm—discriminative RL versus generative SFT—while controlling for data source. This baseline is intended to test whether exposure to correct solutions alone explains the gains, rather than to exhaust all alternative ways of exploiting negative samples.

Together, these ablations aim to distinguish (i) "benefit from judging" from (ii) "benefit from judging first, then generating, " clarifying which component is responsible for improved efficiency and generalization.

4.5 Mechanism Verification

To test style transfer and reduced backtracking, we design:

Perplexity (PPL). PPL is the exponential of a language model's average negative log-likelihood over a sequence; it measures how "surprised" or "confused" the model is, and lower values indicate the model is more familiar with the sequence content. Changes in perplexity can reflect the extent to which a model is familiar with a particular linguistic style. Using the Base SFT model as a fixed evaluator, we compute changes in the PPL of outputs produced under different training methods; these changes then reflect stylistic shifts during training. During judging stage training of JudgeRLVR, we sample model outputs at each step (10 samples per step) for Vanilla RLVR and JudgeRLVR, and compute the average PPL of the Base SFT model on these outputs.

Transition-word statistics. Behaviors such as explicit backtracking and reflection in model outputs are typically manifested through the use of transition-words. Therefore, by counting the number and frequency of such discourse markers in outputs over the course of training, these statistics can indicate, to some extent, that backtracking is decreasing. During the generating stage of JudgeRLVR and the corresponding training steps of Vanilla RLVR, we sample 1000 outputs per step and count occurrences and frequencies of contrast/backtracking markers:

```
but, however, though, although, yet,
still, nevertheless, nonetheless,
instead, conversely, whereas, while,
actually, wait.
```

The above experiments are used to demonstrate shifts in style and backtracking. Combined with improvements in model performance, they can provide a comprehensive indication that these shifts are moving in a favorable direction.

5 Results

5.1 Main Results

Table 1 compares Base SFT, Vanilla RLVR, and JudgeRLVR. Overall, JudgeRLVR achieves higher or comparable accuracy while producing substantially shorter generations on most benchmarks.

In-domain math. On AIME24/25, HMMT, and BeyondAIME, JudgeRLVR consistently improves accuracy over Vanilla RLVR while significantly

Table 2: Ablation: JudgeRLVR (Seq) vs Judge Only and Mixed Strategy (Overall Avg). Δ is vs JudgeRLVR; Δ% is the mean of per-benchmark relative length changes. Full per-benchmark results in Appendix Table 5.

	JudgeRLVR (Seq)		Judge Only				Mixed Strategy			
Split	**Acc↑**	**Len↓**	**Acc↑**	**Δ (pp)**	**Len↓**	**Δ (%)**	**Acc↑**	**Δ (pp)**	**Len↓**	**Δ (%)**
Qwen3-30B-A3B (MoE, 30B total / 3B active)										
In-Domain (Math)	**79.8**	**14.8k**	75.9	-3.9	22.4k	+92	79.7	-0.1	16.4k	+9.5
Out-of-Domain	**78.1**	9.1k	77.8	-0.3	**8.5k**	-15	75.0	-3.1	12.4k	+52
Qwen3-8B (Dense, 8B)										
In-Domain (Math)	**66.3**	**15.7k**	63.3	-3.0	20.2k	+29	66.0	-0.3	17.0k	+8.3
Out-of-Domain	**74.3**	7.0k	73.9	-0.4	**6.9k**	-1	71.8	-2.5	9.5k	+36

Table 3: SFT-then-RLVR vs. JudgeRLVR (Overall Avg). Δ against Vanilla RLVR; Δ% is the mean of per-benchmark relative length changes. Full per-benchmark results in Appendix Table 6.

	Vanilla RLVR		SFT-then-RLVR				JudgeRLVR (Seq)			
Split	**Acc↑**	**Len↓**	**Acc↑**	**Δ (pp)**	**Len↓**	**Δ (%)**	**Acc↑**	**Δ (pp)**	**Len↓**	**Δ (%)**
Qwen3-30B-A3B (MoE, 30B total / 3B active)										
In-Domain (Math)	76.1	23.4k	77.3	+1.2	20.2k	-14	**79.8**	+3.7	**14.8k**	-42
Out-of-Domain	73.7	9.8k	74.9	+1.2	9.4k	-4	**78.1**	+4.5	**9.1k**	-3.9
Qwen3-8B (Dense, 8B)										
In-Domain (Math)	63.4	22.1k	64.2	+0.8	19.5k	-12	**66.3**	+2.9	**15.7k**	-29
Out-of-Domain	71.0	7.6k	71.9	+0.9	7.3k	-4	**74.3**	+3.3	**7.0k**	-8

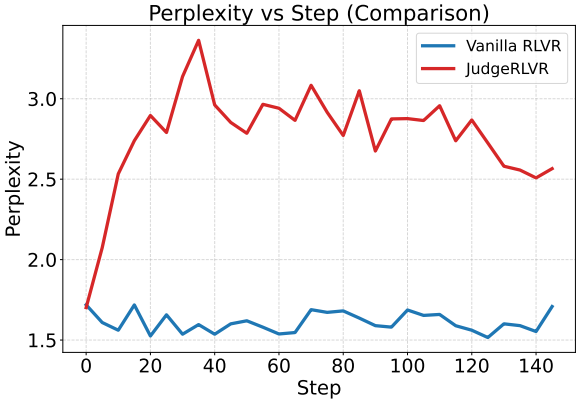

Figure 3: Base SFT perplexity (PPL) evaluated on sampled outputs along training steps. Vanilla RLVR stays stylistically close to Base SFT (flat PPL), while JudgeRLVR (judging stage) exhibits increasing PPL, indicating style transfer.

reducing generation length, with larger gains on harder benchmarks (HMMT, BeyondAIME) that require longer reasoning chains. On the nearly saturated MATH500, accuracy is comparable but length drops sharply, indicating that JudgeRLVR primarily compresses redundant reasoning when headroom for accuracy is limited.

Out-of-domain. On GPQA Diamond, LiveCodeBenchv6, and MMLU-Redux, JudgeRLVR improves both accuracy and efficiency, suggesting that the discriminative prior transfers beyond math. On IFEval and ZebraLogic, we observe higher accuracy but longer outputs; however, certain ablations (e.g., Judge Only on ZebraLogic) achieve comparable accuracy with shorter outputs (Appendix Table 5), suggesting this partly reflects JudgeRLVR's specific accuracy–length trade-off rather than an intrinsic task requirement.

Consistency across architectures. On the dense model (Qwen3-8B), JudgeRLVR exhibits the same trends: **+2.9 pp** in-domain accuracy with **29%** mean per-benchmark length reduction, and **+3.3 pp** out-of-domain accuracy (Table 1, bottom). Per-benchmark patterns mirror those on the MoE model, despite differences in architecture, scale, and initialization source.

5.2 Ablation Results

Table 2 reports ablations testing whether judging stage alone is sufficient and whether sequential staging is essential.

Judge Only. On in-domain math, Judge Only degrades accuracy and increases length compared with JudgeRLVR, indicating that discriminative capability alone does not yield concise generation.

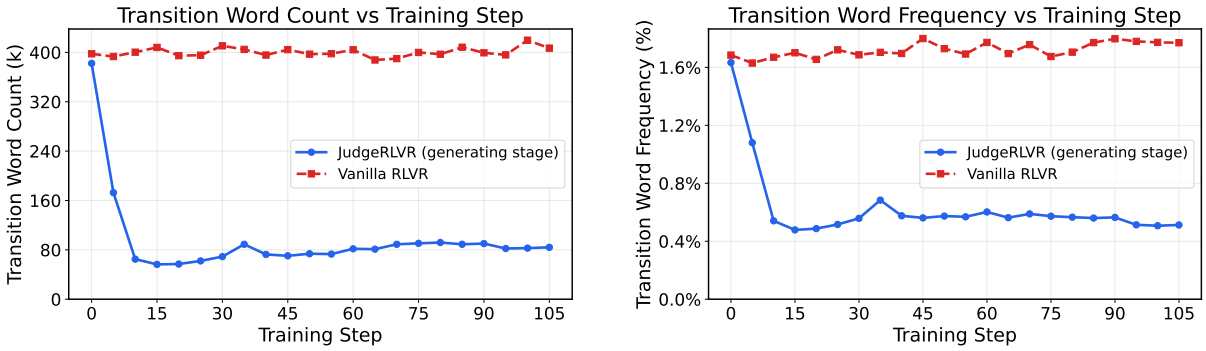

Figure 4: Counts (left) and frequencies (right) of transition/backtracking markers (e.g., *but*, *however*, *wait*) in sampled outputs across training steps for both JudgeRLVR (generating stage) and Vanilla RLVR (corresponding steps). JudgeRLVR exhibits a substantially steeper decline in backtracking markers, while Vanilla RLVR remains relatively flat or increases slightly.

On some out-of-domain tasks (e.g., GPQA, ZebraLogic), Judge Only achieves comparable accuracy with shorter outputs, but this benefit is task-dependent and unstable (Appendix Table 5).

Mixed Strategy. Mixed Strategy occasionally matches JudgeRLVR on individual tasks (e.g., AIME25, MATH500) but drops substantially on others (IFEval, LiveCodeBenchv6) with longer outputs, consistent with interference from simultaneously optimizing two objectives in one phase.

SFT-then-RLVR. Replacing Stage 1's discriminative RL with generative SFT (Table 3) yields a modest **+1.2 pp** accuracy gain over Vanilla RLVR with **14%** mean per-benchmark length reduction. JudgeRLVR further improves by **+2.5 pp** in-domain and **+3.2 pp** out-of-domain with far greater length reduction (**-42%** vs. **-14%**). Since both methods use the same positive training data, this gap is consistent with the discriminative RL objective being a key driver.

5.3 Mechanism Verification

Style transfer measured by PPL under Base SFT. Figure 3 reports the Base SFT model's average perplexity on sampled outputs across training steps. For Vanilla RLVR, the average PPL remains largely unchanged, suggesting the output distribution stays stylistically close to Base SFT. In contrast, during JudgeRLVR judging stage, Base SFT PPL increases noticeably, indicating a systematic style shift induced by judge training.

Reduced explicit backtracking measured by transition words. Figure 4 shows that during the generating stage, JudgeRLVR's absolute count and frequency of transition/backtracking words decrease substantially over training, whereas Vanilla RLVR remains relatively stable or slightly increases over the corresponding steps. This divergence is consistent with the reduction in explicit backtracking being specific to JudgeRLVR rather than a general effect of continued RL training, supporting the hypothesis that the discriminative prior from the judging stage helps suppress unproductive self-correction during generation.

6 Conclusion

In this paper, we proposed JudgeRLVR, a two-stage training paradigm that acts as a precursor to efficient reasoning. By explicitly training the model to discriminate solution validity before generating, we achieved a superior quality–efficiency trade-off. Experiments on both MoE (Qwen3-30B-A3B) and dense (Qwen3-8B) architectures show that JudgeRLVR improves in-domain math accuracy with up to 42% mean per-benchmark length reduction, with notable generalization to out-of-domain tasks. A controlled SFT-then-RLVR ablation further demonstrates that the gains stem from the discriminative RL objective, not merely from warm-start effects.

Many assume that judging an answer is the easy part, because verification seems to come with a checklist that generation does not. Our study points to a sharper truth: on identical math problems with the same model, judging yields greater improvement than solving alone. Training the judge makes the model internalize what a good line of thought looks like, and later generation can cross its own limits with fewer false starts. In this sense, to generate is simply to judge so well that the correct answer is the only thing left to say.

Limitations

Despite achieving a favorable quality–efficiency trade-off on RLVR and showing strong generalization, our work has several limitations that merit further investigation:

Limited training domains Our experiments only train on mathematics-domain data. Whether the same training strategy and observed gains transfer to other domains (e.g., coding, scientific QA, open-domain reasoning) remains to be validated.

Coarse-grained judge signals We employ an outcome-level judge on the final solution response with a binary correct/incorrect decision. This supervision may be insufficient to capture fine-grained reasoning errors (e.g., identifying the exact incorrect step), and more granular judge feedback could further improve learning and diagnostic ability.

References

Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. 2025. Matharena: Evaluating llms on uncontaminated math competitions.

[ByteDance-Seed]. 2025. Beyondaime: Advancing math reasoning evaluation beyond high school olympiads. https://huggingface.co/datasets/ByteDance-Seed/BeyondAIME.

Hongyi James Cai and et al. 2025. How much backtracking is enough? exploring the interplay of sft and rl in enhancing llm reasoning.

Michelene T. H. Chi, Paul J. Feltovich, and Robert Glaser. 1981. Categorization and representation of physics problems by experts and novices. *Cognitive Science*, 5(2):121–152.

AI DataLearner. 2025. Reinforcement learning with verifiable rewards (rlvr): Why the focus of llm training is shifting in 2025. *DataLearner Official Blog*.

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Aryo P. Gema, Joshua O. J. Leang, Giwon Hong, Alessio Devoto, Alberto C. M. Mancino, Rohit Saxena, Xuanli He, Yu Zhao, Xiaotang Du, Mohammad R. G. Madani, Claire Barale, Robert McHardy, Joshua Harris, Jean Kaddour, Emile Van Krieken, and Pasquale Minervini. 2025. Are we done with mmlu? In *Proceedings of NAACL 2025*.

Jian Guo, Jie Tang, and Xiaolin Li. 2025a. Deepseek-r1: Scaling rlvr for mathematical reasoning.

Yuchen Guo, Xin Wang, Yifan Zhang, and Pengfei Liu. 2025b. Discriminative finetuning of generative large language models without reward models and preference data.

Xu Han, Yan Wang, Zonghan Li, and Chengqi Zhang. 2024. Uncertainty-based verifiers for llm reasoning outputs.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the MATH dataset. *Advances in Neural Information Processing Systems (NeurIPS)*.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. Livecodebench: Holistic and contamination free evaluation of large language models for code.

Nathan Lambert, Noah A. Smith, and Hannaneh Hajishirzi. 2024. Scalable verifiable reward learning for large language models.

Bill Yuchen Lin, Ronan Le Bras, Kyle Richardson, Ashish Sabharwal, Radha Poovendran, Peter Clark, and Yejin Choi. 2025. Zebralogic: On the scaling limits of llms for logical reasoning.

Ilya Loshchilov and Frank Hutter. 2018. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*.

Moonshot AI Team. 2025. Kimi k1.5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*.

David Rein, Betty Li Hou, Asa C. Stickland, Jackson Petty, Richard Y. Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2023. Gpqa: A graduate-level google-proof q&a benchmark.

Yuxin Shao, Tianyu Liu, Minghao Chen, and Shuohang Li. 2025. Spurious rewards: Rethinking training signals in rlvr.

Wei Shen, Jia Li, Hongyu Zhang, and Zhe Wang. 2025. Llm reasoning engine: Specialized training for enhanced mathematical reasoning. In *Proceedings of the 4th International Workshop on Knowledge-Augmented Methods for Natural Language Processing (KnowledgeNLP'25)*, pages 118–128. Association for Computational Linguistics.

Luis Shimabucoro, Rutuja Chavhan, Timothy Hospedales, and Henry Gouk. 2024. Evaluating the evaluators: Are validation methods for few-shot learning fit for purpose? *Transactions on Machine Learning Research*, pages 1–13.

Zhiqing Sun, Yang Liu, Danqi Chen, and Wenhui Li. 2024. Skywork-math: Scaling mathematical reasoning with corpus fine-tuning.

Xinyi Tang, Jiaqi Chen, Bing Zhang, and Zhiyuan Liu. 2025. Unifiedreward-think: A comprehensive evaluator for reasoning quality.

DeepSeek Team, Feng Liu, Hao Wang, and Bo Zhang. 2024. Deepseek-math: Enhancing mathematical reasoning via rlvr. In *Proceedings of the 38th International Conference on Machine Learning (ICML'24)*, pages 2345–2356. PMLR.

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

LLM-Core-Team Xiaomi. 2025. Mimo: Unlocking the reasoning potential of language model – from pretraining to posttraining. *Preprint*, arXiv:2505.07608.

Yilun Xu, Tuhin Chakraborty, Shubham Sharma, Lucas Nunes, Emre Kıcıman, Shan Lu, and Rakesh Chandra. 2025. Direct reasoning optimization: Llms can reward and refine their own reasoning for open-ended tasks.

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

Xiang Yue, Zheqi Chen, and Gao Huang. 2025. Does reinforcement learning with verifiable rewards elicit novel reasoning abilities?

Yifan Zhang and Team Math-AI. 2024. American invitational mathematics examination (aime) 2024.

Yifan Zhang and Team Math-AI. 2025. American invitational mathematics examination (aime) 2025.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. Instruction-following evaluation for large language models.

Mingsheng Zhou, Jun Li, Xiaoyu Wang, and Lingling Zhang. 2025. Jiuzhang3.0: Efficient mathematical reasoning via synthetic data pre-training.

Xinyu Zhu, Mengzhou Xia, Zhepei Wei, Wei-Lin Chen, Danqi Chen, and Yu Meng. 2025. The surprising effectiveness of negative reinforcement in llm reasoning.

A Prompt Templates of Judge and Generate

Judging Prompt

```
Given the question:
{question}
And the proposed answer:
{solution response}
Analyze concisely whether the answer is correct. After your analysis, put only your final judgment in \boxed{} as 1 for correct and 0 for incorrect.
```

Generating and Rollout Prompt

```
{question}
Let's think step by step and output the final answer within \boxed{}.
```

B Case Study

Full output of Figure 1

Problem Statement

Convert the point $(0, 3)$ in rectangular coordinates to polar coordinates. Enter your answer in the form (r, θ), where $r > 0$ and $0 \leq \theta < 2\pi$.

Vanilla RLVR

<think>

I need to convert the point (0,3) from rectangular coordinates to polar coordinates. The rectangular coordinates are given as (x,y) = (0,3), and I need to find the polar coordinates (r, θ), where r is the distance from the origin, and θ is the angle from the positive x-axis.

First, recall the formulas for conversion. The radius r is given by r = $\sqrt{(x^2 + y^2)}$. Plugging in the values, x is 0 and y is 3, so r = $\sqrt{(0^2 + 3^2)}$ = $\sqrt{(0 + 9)}$ = $\sqrt{9}$ = 3. So r is 3.

Now, for the angle θ, it's given by $\theta = \tan^{-1}(y/x)$. But here, x is 0, and y is 3. So if I plug in, $\tan^{-1}(3/0)$, which is undefined because division by zero. That makes sense because when x is 0, the point is on the y-axis.

I know that when a point is on the positive y-axis, the angle θ should be $\pi/2$ radians, or 90 degrees. Since the problem asks for θ in radians between 0 and 2π, I should use radians.

So, $\theta = \pi/2$.

Let me confirm. If r is 3, and θ is $\pi/2$, then the rectangular coordinates should be $x = r\cos\theta = 3 * \cos(\pi/2) = 3 * 0 = 0$, and $y = r\sin\theta = 3 * \sin(\pi/2) = 3 * 1 = 3$. Perfect, that matches.

The problem says r > 0, which is satisfied since r=3>0, and θ between 0 and 2π, $\pi/2$ is within that range.

I recall that sometimes people might think of θ as $-\pi/2$ or $3\pi/2$, but that would give a negative y-value, which is not the case here. Since y is positive, it has to be in the first or second quadrant, but since x is 0, it's specifically on the positive y-axis, so $\theta = \pi/2$.

If θ were $3\pi/2$, that would be the negative y-axis, where y is negative, but here y is positive, so it's not that.

So, I think $(3, \pi/2)$ is the correct polar coordinate.

The answer should be in the form (r, θ), so (3, $\pi/2$).

I should write it as a pair. And the problem says to box the final answer.

So, I think that's it.

Just to be thorough, is there any other representation? For example, could r be negative? But the problem specifies r > 0, so no. And θ is usually taken in $[0, 2\pi)$, so $\pi/2$ is fine.

Sometimes people use degrees, but here it's specified to use radians, and the range is given, so radians are implied.

The problem doesn't specify units, but in mathematics, especially with π, it's standard to use radians.

So, I'm confident.

</think>

First, I need to convert the point $(0, 3)$ from rectangular coordinates to polar coordinates (r, θ) with $r > 0$ and $0 \leq \theta < 2\pi$, use the conversion formulas.

The radius r is given by:

$$r = \sqrt{x^2 + y^2} = \sqrt{0^2 + 3^2} = \sqrt{9} = 3.$$

The angle θ is found using $\theta = \tan^{-1}(y/x)$. However, since $x = 0$ and $y = 3 > 0$, the point lies on the positive y-axis. Thus, $\theta = \pi/2$.

Verify the solution:

- $x = r\cos\theta = 3\cos(\pi/2) = 3 \cdot 0 = 0$,
- $y = r\sin\theta = 3\sin(\pi/2) = 3 \cdot 1 = 3$,

which matches the original point.

The conditions are satisfied: $r = 3 > 0$ and $\theta = \pi/2$ is in $[0, 2\pi)$.

Thus, the polar coordinates are $(3, \pi/2)$.]

The final answer is:

$$\boxed{\left(3, \frac{\pi}{2}\right)}$$

JudgeRLVR

<think>

We need to convert (0,3) to polar coordinates. r = sqrt(x^2 + y^2) = sqrt(0+9)=3. Theta is angle from positive x-axis to point (0,3). That's 90 degrees = pi/2 rad. Since x=0, y>0, theta = $\pi/2$. So answer (3, $\pi/2$). Ensure within 0 to 2π. So final answer $\boxed{(3, \frac{\pi}{2})}$.

</think>

Table 4: Evaluation Benchmarks

Benchmark	Metric
AIME24	Acc avg@10
AIME25	Acc avg@10
MATH500	Acc
HMMT_feb_2025	Acc avg@10
BeyondAIME	Acc avg@8
GPQA Diamond	Exact Match avg@5
IFEval	Prompt Strict
LiveCodeBenchv6	Pass@1
MMLU-Redux	Exact Match
ZebraLogic	Acc

First, I need to calculate r.

The point $(0, 3)$ has

$$r = \sqrt{0^2 + 3^2} = 3$$

and lies on the positive y-axis, so the angle from the positive x-axis is

$$\theta = \frac{\pi}{2}.$$

Thus the polar coordinates are $(3, \frac{\pi}{2})$.

The final answer is:

$$\boxed{(3, \frac{\pi}{2})}$$

C Evaluate Details

Table 4 shows the metric setup of evaluation benchmarks.

D Full Per-Benchmark Results

Table 5: Full per-benchmark ablation: JudgeRLVR (Seq) vs Judge Only and Mixed Strategy. Δ is vs JudgeRLVR; Overall Δ% is the mean of per-benchmark relative length changes.

	JudgeRLVR (Seq)		**Judge Only**				**Mixed Strategy**			
Benchmark	**Acc↑**	**Len↓**	**Acc↑**	**Δ (pp)**	**Len↓**	**Δ (%)**	**Acc↑**	**Δ (pp)**	**Len↓**	**Δ (%)**
Qwen3-30B-A3B (MoE, 30B total / 3B active)										
In-Domain (Math)										
AIME24	**89.0**	**12.9k**	87.0	-2.0	22.4k	+74	88.3	-0.7	12.8k	-0.8
AIME25	78.7	**17.2k**	76.3	-2.4	28.5k	+66	**79.7**	+1.0	18.2k	+6.0
MATH500	97.2	**2.0k**	97.2	+0.0	7.4k	+266	**98.0**	+0.8	2.2k	+7.9
HMMT_feb_2025	**70.0**	20.4k	61.5	-8.5	**17.9k**	-12	69.0	-1.0	25.9k	+27
BeyondAIME	**63.9**	**21.3k**	57.3	-6.6	35.8k	+68	63.4	-0.5	22.9k	+7.3
Overall Avg	79.8	14.8k	75.9	-3.9	22.4k	+92	79.7	-0.1	16.4k	+9.5
Out-of-Domain										
GPQA Diamond	66.4	11.2k	**66.9**	+0.5	**7.9k**	-30	63.7	-2.7	16.1k	+44
IFEval	**86.4**	4.6k	85.9	-0.5	**4.2k**	-9.0	80.4	-6.0	7.9k	+72
LiveCodeBenchv6	**63.9**	**18.7k**	61.4	-2.5	21.3k	+14	56.4	-7.5	26.7k	+43
MMLU Redux	87.0	1.5k	**87.4**	+0.4	**1.0k**	-31	**87.1**	+0.1	3.3k	+119
ZebraLogic	86.8	9.6k	87.3	+0.5	8.0k	-17	**87.4**	+0.6	**7.9k**	-19
Overall Avg	78.1	9.1k	77.8	-0.3	8.5k	-15	75.0	-3.1	12.4k	+52
Qwen3-8B (Dense, 8B)										
In-Domain (Math)										
AIME24	**75.3**	**15.0k**	73.5	-1.8	21.2k	+41	74.7	-0.6	15.2k	+1.3
AIME25	64.0	**17.0k**	62.0	-2.0	24.5k	+44	**64.3**	+0.3	17.5k	+2.9
MATH500	96.8	**2.5k**	96.8	+0.0	7.2k	+188	**97.2**	+0.4	2.7k	+8.0
HMMT_feb_2025	**56.5**	21.0k	50.0	-6.5	**17.5k**	-17	55.5	-1.0	25.5k	+21
BeyondAIME	**39.0**	**23.0k**	34.0	-5.0	30.5k	+33	38.5	-0.5	24.0k	+4.3
Overall Avg	66.3	15.7k	63.3	-3.0	20.2k	+29	66.0	-0.3	17.0k	+8.3
Out-of-Domain										
GPQA Diamond	64.2	8.5k	**64.5**	+0.3	**7.8k**	-8	61.5	-2.7	14.0k	+65
IFEval	**78.5**	1.4k	78.0	-0.5	**1.3k**	-7	74.0	-4.5	2.2k	+57
LiveCodeBenchv6	**54.8**	**16.8k**	52.5	-2.3	18.5k	+10	49.0	-5.8	23.0k	+37
MMLU Redux	84.2	1.5k	**84.5**	+0.3	**1.0k**	-33	**84.5**	+0.3	3.0k	+100
ZebraLogic	89.8	7.0k	90.0	+0.2	5.8k	-17	**90.2**	+0.4	**5.5k**	-21
Overall Avg	74.3	7.0k	73.9	-0.4	6.9k	-1	71.8	-2.5	9.5k	+36

Table 6: Full per-benchmark: SFT-then-RLVR vs. JudgeRLVR. Δ against Vanilla RLVR; Overall Δ% is the mean of per-benchmark relative length changes.

	Vanilla RLVR		SFT-then-RLVR				JudgeRLVR (Seq)			
Benchmark	**Acc↑**	**Len↓**	**Acc↑**	**Δ (pp)**	**Len↓**	**Δ (%)**	**Acc↑**	**Δ (pp)**	**Len↓**	**Δ (%)**
Qwen3-30B-A3B (MoE, 30B total / 3B active)										
In-Domain (Math)										
AIME24	86.3	21.8k	87.3	+1.0	18.5k	-15	**89.0**	+2.7	**12.9k**	-41
AIME25	77.0	27.2k	77.7	+0.7	23.4k	-14	**78.7**	+1.7	**17.2k**	-37
MATH500	**98.0**	6.8k	97.8	-0.2	4.5k	-34	97.2	-0.8	**2.0k**	-71
HMMT_feb_2025	60.8	26.9k	63.5	+2.7	24.1k	-10	**70.0**	+9.2	**20.4k**	-24
BeyondAIME	58.3	34.3k	60.1	+1.8	30.6k	-11	**63.9**	+5.6	**21.3k**	-38
Overall Avg	76.1	23.4k	77.3	+1.2	20.2k	-14	**79.8**	+3.7	**14.8k**	-42
Out-of-Domain										
GPQA Diamond	61.2	12.1k	62.8	+1.6	11.6k	-4	**66.4**	+5.2	**11.2k**	-7.5
IFEval	79.9	4.1k	81.4	+1.5	4.0k	-2	**86.4**	+6.5	4.6k	+12
LiveCodeBenchv6	58.2	22.8k	59.6	+1.4	21.5k	-6	**63.9**	+5.7	**18.7k**	-18
MMLU Redux	85.4	2.1k	86.2	+0.8	1.9k	-10	**87.0**	+1.6	**1.5k**	-28
ZebraLogic	83.6	7.9k	84.4	+0.8	8.1k	+3	**86.8**	+3.2	9.6k	+22
Overall Avg	73.7	9.8k	74.9	+1.2	9.4k	-4	**78.1**	+4.5	**9.1k**	-3.9
Qwen3-8B (Dense, 8B)										
In-Domain (Math)										
AIME24	73.0	21.5k	73.7	+0.7	19.0k	-12	**75.3**	+2.3	**15.0k**	-30
AIME25	62.0	24.0k	62.7	+0.7	21.5k	-10	**64.0**	+2.0	**17.0k**	-29
MATH500	**97.4**	7.6k	97.2	-0.2	5.0k	-34	96.8	-0.6	**2.5k**	-67
HMMT_feb_2025	49.2	27.0k	51.0	+1.8	24.5k	-9	**56.5**	+7.3	**21.0k**	-22
BeyondAIME	35.3	30.5k	36.5	+1.2	27.5k	-10	**39.0**	+3.7	**23.0k**	-25
Overall Avg	63.4	22.1k	64.2	+0.8	19.5k	-12	**66.3**	+2.9	**15.7k**	-29
Out-of-Domain										
GPQA Diamond	60.6	9.2k	61.8	+1.2	8.8k	-4	**64.2**	+3.6	**8.5k**	-8
IFEval	73.2	1.2k	74.4	+1.2	1.2k	0	**78.5**	+5.3	1.4k	+17
LiveCodeBenchv6	50.6	19.5k	51.5	+0.9	18.5k	-5	**54.8**	+4.2	**16.8k**	-14
MMLU Redux	82.8	2.0k	83.5	+0.7	1.8k	-10	**84.2**	+1.4	**1.5k**	-25
ZebraLogic	87.8	6.2k	88.5	+0.7	6.3k	+2	**89.8**	+2.0	7.0k	+13
Overall Avg	71.0	7.6k	71.9	+0.9	7.3k	-4	**74.3**	+3.3	**7.0k**	-8