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.

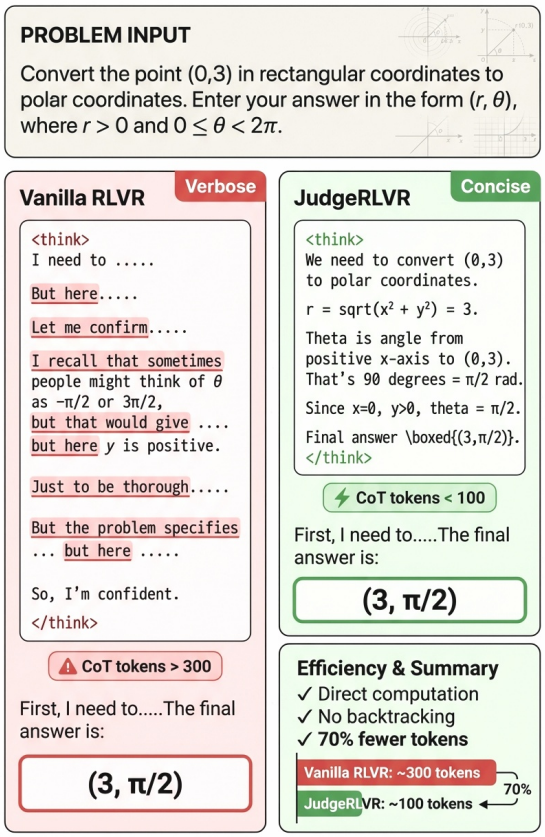

Figure 1: Qualitative comparison of the reasoning process induced by Vanilla RLVR and JudgeRLVR. **Left**: Vanilla RLVR induces lengthy and redundant reasoning. **Right**: JudgeRLVR implicitly regularizes the reasoning process, leading to a direct and concise solution path.

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., 2025b; 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.

While RLVR optimizes performance, it also implicitly shapes the model's thinking process in pursuit of the correct answer. As we know, when trained primarily with a final-answer reward, LLMs tend to drift toward a generative search style: enumerating tentative branches, revising intermediate steps, and performing explicit self-correction to secure the reward (Shao et al., 2024). Even when the reasoning steps are already correct, the model lacks the ability to make a definitive judgment; in-

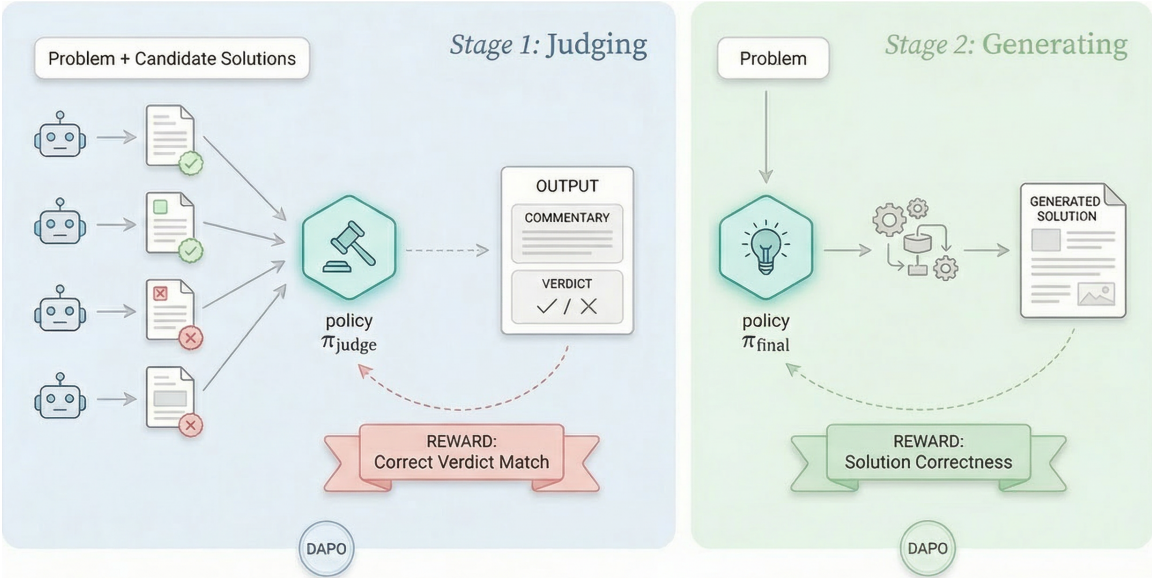

Figure 2: The two-stage JudgeRLVR training paradigm

stead, it becomes trapped in continuous backtracking and re-evaluation. Consequently, the resulting CoT trajectories are often long and exhibit frequent backtracking (Cai et al., 2025), which incurs an expensive computational cost. As illustrated in Figure 1, the CoT of Vanilla RLVR is peppered with cue words like "let me confirm" and "but" that drive continuous exploration. Because the model fails to make a timely judgment to stop even when the answer is reached, it generates long trajectories with low information density (see Appendix B), leading to significant computational waste. Existing methods such as Kimi K1.5 (Kimi Team et al., 2025) and DAPO (Yu et al., 2025) introduce length penalties to reduce tokens and stabilize training, but often come with accuracy drops.

In our view, Vanilla RLVR lacks the ability to discriminate valid reasoning, a deficiency that cannot be resolved simply by introducing a reward like a length penalty. Cognitive science suggests that expert reasoning relies on *early discrimination and pruning*, i.e., recognizing correct paths or filtering unpromising paths before expanding them, rather than exhaustive search (Chi et al., 1981). This motivates our judge-then-generate view of efficient reasoning: a model that first learns to judge whether a candidate solution is valid may internalize an error-aware prior that makes its later reasoning process more selective and direct, without relying on explicit length control.

Based on the motivation above, we propose **JudgeRLVR**, a two-stage *judge-then-generate* training 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 to build the model's intrinsic sensitivity to error, instilling a preference for direct and effective reasoning processes. In the generation training stage, building upon the judge capability, we follow the training methodology of Vanilla RLVR. By allowing a model equipped with this discriminative ability to explore correct answers, it can successfully generate concise reasoning processes. Notably, we impose no explicit length or quality penalties on the generated reasoning process. This means that any reduction in verbosity can be interpreted as arising from the discriminative prior acquired in the judging stage.

In our experiments, compared to Vanilla RLVR using the same training data, JudgeRLVR achieves a quality–efficiency trade-off on both Qwen3-30B-A3B (MoE) (Yang et al., 2025) and Qwen3-8B (dense) (Yang et al., 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 a high-quality reasoning process 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 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) improves reasoning by optimizing automatically checkable answers. DeepSeekMath, DeepSeek-R1, Kimi k1.5, and DAPO progressively scaled outcome-reward RL to stronger models (Shao et al., 2024; DeepSeek-AI et al., 2025; Kimi Team et al., 2025; Yu et al., 2025). However, outcome rewards can exploit spurious signals and may mainly reweight paths already present in the base model (Shao et al., 2025; Yue et al., 2025). Recent studies further examine whether verifiable rewards induce robust compositional reasoning and ensure causally important reasoning steps (Barzilai et al., 2026; Yu et al., 2026). Process-supervised RL has also been explored, e.g., agentic RAG methods that reward intermediate retrieval actions rather than only the final answer (Zhang et al., 2025). JudgeRLVR shares the goal of reducing dependence on sparse final-answer rewards but studies a different mechanism: using discriminative judge training as initialization for outcome-reward generation.

2.2 Mathematical Reasoning in Large Language Models

Mathematical reasoning benchmarks drive LLM reasoning research, spanning verifier training, chain-of-thought prompting, and self-consistency decoding (Hendrycks et al., 2021; Cobbe et al., 2021; Wei et al., 2022; Wang et al., 2023b). Recent systems improve accuracy via synthetic data and RL (Shao et al., 2024; Zhou et al., 2024; Zeng et al., 2024; Chen and Lin, 2025), and offline RL has also been studied as a route to stronger math reasoning (Lu et al., 2026). These works primarily target correctness via data or reward optimization. JudgeRLVR is complementary: it first trains the model to distinguish correct from incorrect solution responses, then transfers that discriminative prior into generation.

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

Judging generated answers has served as a verifier, reward signal, or evaluator. Process-supervision trains step-level reward models, and Math-Shepherd constructs process-wise supervision without human annotations (Cobbe et al., 2021; Lightman et al., 2023; Wang et al., 2023a). Recent work further shows that incorporating chain-of-thought reasoning into reward modeling improves reward accuracy and robustness (Wang et al., 2025), and that lightweight uncertainty heads can efficiently verify individual reasoning steps via internal model states (Ni et al., 2026). LLM-as-a-judge work studies scalable evaluators and rubric-conditioned feedback (Zheng et al., 2023; Zhu et al., 2025a; Kim et al., 2024), while discriminative fine-tuning contrasts positive and negative outputs without a separate reward model (Guo et al., 2025). Recent work also uses LLM-as-a-judge signals for reinforcement-learning-based distillation (Shen et al., 2026). Closest to our setting, S2R teaches models to self-verify and self-correct during inference via outcome- and process-level RL (Ma et al., 2025). Existing approaches integrate judgment either as an explicit, online component of the reasoning process (e.g., interleaved verification steps) or as a reward signal during training. JudgeRLVR differs: it trains discriminative judgment as a standalone stage and transfers the weights to initialize generative RLVR, so that error-awareness is internalized as an implicit prior rather than added at inference time.

3 Method

3.1 Overview

The core intuition of **JudgeRLVR** is that discriminative competence is a prerequisite for efficient generation. As shown in Figure 2, JudgeRLVR divides the training of a single model π_θ into two sequential stages: (1) a *judging stage* that trains the model to evaluate candidate solutions as correct or incorrect, building discriminative error awareness; and (2) a *generating stage* that, starting from the judging-stage checkpoint, trains the model to generate solutions itself, translating the discriminative prior into concise generation. Both stages use the same RL algorithm (DAPO) and training data source; no length penalty or process reward is applied.

3.2 Stage 1: Judging Stage

Task formulation. In this stage, the model π_θ is trained as a discriminative judge via RL. The input consists of a query $x \in \mathcal{X}$ paired with a candidate solution response s (generated by external models). The ground-truth correctness label of s is determined by comparing the extracted answer against the gold answer:

$$g(x, s) = \mathbb{I}\big(a(s) = a^\star(x)\big), \tag{1}$$

where $a(s)$ is the final answer extracted from s using a deterministic parser and $a^\star(x)$ is the gold answer for query x.

Given (x, s), the policy generates a judge commentary c followed by a binary verdict $v \in \{0, 1\}$ (0 = incorrect, 1 = correct):

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

Reward. The reward is whether the predicted verdict v matches the ground-truth label:

$$r^{\text{judge}} = \mathbb{I}\big(v = g(x, s)\big). \tag{3}$$

The commentary c serves as a reasoning prefix before the verdict and is optimized by the same policy-gradient signal—it receives the same episode reward as v. This encourages the model to develop structured analysis that supports accurate judgment.

Data construction. We construct the judging dataset $\mathcal{D}_{\text{judge}}$ consisting of triplets (x, s, g):

- **Response generation (rollout).** For each query x, we sample a set of candidate solution responses $\{s_1, \ldots, s_n\}$ from several models, and obtain $g(x, s_i)$ by comparing $a(s_i)$ 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 ($g = 1$) and negatives ($g = 0$), avoiding majority-class bias.

We judge these relatively clean candidate responses rather than very long raw rollouts, so that the reward mainly targets solution validity instead of distracting formatting or exploration artifacts.

3.3 Stage 2: Generating Stage

Task formulation. In this stage, the model π_θ is trained as a solution generator. The policy is initialized from the Stage 1 checkpoint, and the input format switches from the judge template to a standard generation prompt—only the query x is provided, with no candidate solution.

Given a query x, the policy generates a complete solution response y (including intermediate reasoning and the final answer):

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

Reward. The reward is the sparse binary correctness signal based solely on the final answer:

$$r^{\text{gen}} = \mathbb{I}\big(a(y) = a^\star(x)\big), \tag{5}$$

where $a(y)$ extracts the final answer from the generated solution y, and $a^\star(x)$ is the gold answer. This is identical to the standard Vanilla RLVR reward—no length penalty or process reward is added.

Connection to Stage 1. The only difference from Vanilla RLVR is the initialization: the policy starts from the Stage 1 judge checkpoint rather than from the base model. Our hypothesis is that the discriminative prior acquired in Stage 1 enables the model to recognize correct reasoning paths earlier during generation, leading to more concise outputs without sacrificing accuracy.

3.4 Hypothesized Mechanism and Verification Overview

We hypothesize that JudgeRLVR improves reasoning-pattern quality through two corresponding mechanisms: (i) **style transfer**: the judging stage induces a global shift in the model's reasoning language style, moving away from verbose exploratory patterns; (ii) **reduced backtracking**: the generating stage activates the internalized discriminative prior, reducing explicit self-correction and backtracking in the generated text. We design targeted experiments to test both hypotheses; 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)

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

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 et al., 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 LLM-Core Team et al., 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, 2019) 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 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 (Mathematical Association of America, 2024), AIME25 (Mathematical Association of America, 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., 2025),

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. This setting tests whether judge training alone improves generation, or whether the generating stage is necessary to translate judging competence into better generation, directly probing whether 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. This 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.

SFT-then-RLVR: to isolate whether 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, isolating the Stage 1 paradigm—discriminative RL versus generative SFT—while controlling for data source.

Together, these ablations distinguish "benefit from judging" from "benefit from judging first, then generating."

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; lower values indicate the model is more familiar with the content, so changes in PPL can reflect shifts in 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. Explicit backtracking and reflection in model outputs are typically manifested through transition-words, so counting their occurrences and frequencies over training can indicate whether 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,`

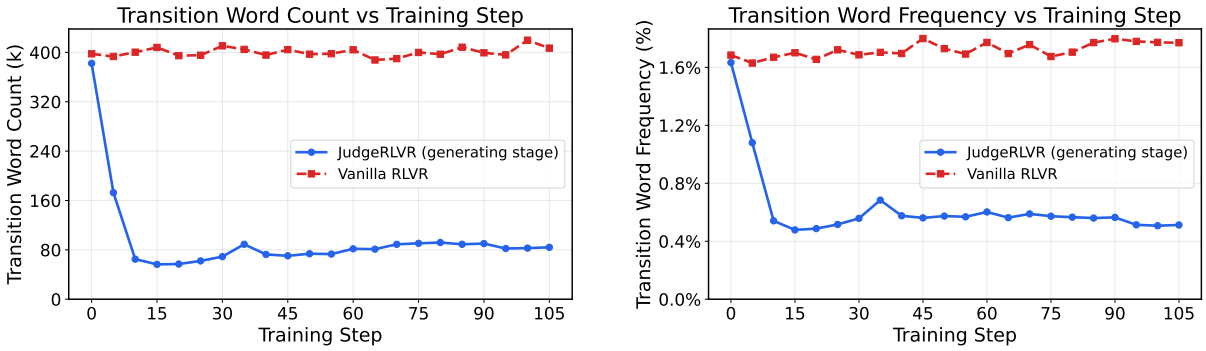

Figure 3: Counts (left) and frequencies (right) of transition/backtracking markers (e.g., *but*, *however*, *wait*) in sampled outputs across training steps for both JudgeRLVR (generating stage) and Vanilla RLVR (corresponding steps). JudgeRLVR exhibits a substantially steeper decline in backtracking markers, while Vanilla RLVR remains relatively flat or increases slightly.

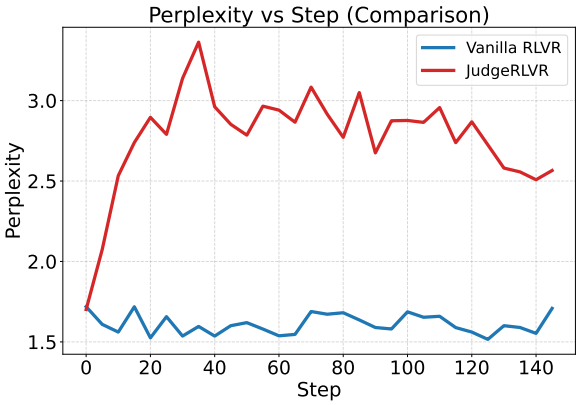

Figure 4: Base SFT perplexity (PPL) evaluated on sampled outputs along training steps. Vanilla RLVR stays stylistically close to Base SFT (flat PPL), while JudgeRLVR (judging stage) exhibits increasing PPL, indicating style transfer.

```
still,    nevertheless,    nonetheless,
instead,  conversely,  whereas,  while,
actually, wait.
```

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 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 the discriminative prior transfers beyond math. On IFEval and ZebraLogic, accuracy is higher but outputs are longer; certain ablations (e.g., Judge Only on ZebraLogic) reach 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. 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).

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

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 4 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 3 shows that during the generating stage, JudgeRLVR's absolute count and frequency of transition/backtracking words decrease substantially, whereas Vanilla RLVR remains relatively stable or slightly increases. 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.

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. *Preprint*, arXiv:2505.23281.

Daniel Barzilai, Yotam Wolf, and Ronen Basri. 2026. When is compositional reasoning learnable from verifiable rewards? *Preprint*, arXiv:2602.07992.

ByteDance-Seed. 2025. Beyondaime: Advancing math reasoning evaluation beyond high school olympiads. Hugging Face dataset.

Hongyi James Cai, Junlin Wang, Xiaoyin Chen, and Bhuwan Dhingra. 2025. How much backtracking is enough? exploring the interplay of SFT and RL in enhancing LLM reasoning. *Preprint*, arXiv:2505.24273.

Shuguang Chen and Guang Lin. 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*, pages 118–128, Albuquerque, New Mexico, USA. Association for Computational Linguistics.

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.

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

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

Aryo Pradipta Gema, Joshua Ong Jun Leang, Giwon Hong, Alessio Devoto, Alberto Carlo Maria Mancino, Rohit Saxena, Xuanli He, Yu Zhao, Xiaotang Du, Mohammad Reza Ghasemi Madani, Claire Barale, Robert McHardy, Joshua Harris, Jean Kaddour, Emile Van Krieken, and Pasquale Minervini. 2025. Are we done with MMLU? In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 5069–5096, Albuquerque, New Mexico. Association for Computational Linguistics.

Siqi Guo, Ilgee Hong, Vicente Balmaseda, Changlong Yu, Liang Qiu, Xin Liu, Haoming Jiang, Tuo Zhao, and Tianbao Yang. 2025. Discriminative finetuning of generative large language models without reward models and human preference data. *Preprint*, arXiv:2502.18679.

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. In *Advances in Neural Information Processing Systems*, volume 34, pages 25731–25746.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2025. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*.

Seungone Kim, Jamin Shin, Yejin Cho, Joel Jang, Shayne Longpre, Hwaran Lee, Sangdoo Yun, Seongjin Shin, Sungdong Kim, James Thorne, and Minjoon Seo. 2024. Prometheus: Inducing fine-grained evaluation capability in language models. In *International Conference on Learning Representations*.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, and 1 others. 2025. Kimi k1.5: Scaling reinforcement learning with llms. *Preprint*, arXiv:2501.12599.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *Preprint*, arXiv:2305.20050.

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. In *Proceedings*

of the 42nd International Conference on Machine Learning, volume 267 of *Proceedings of Machine Learning Research*, pages 37889–37905. PMLR.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Yao Lu, Dengdong Fan, Jianzheng Nie, Fan Xu, Jie Chen, Bin Zhou, and Yonghong Tian. 2026. Pcl-reasoner-v1.5: Advancing math reasoning with offline reinforcement learning. *Preprint*, arXiv:2601.14716.

Ruotian Ma, Peisong Wang, Cheng Liu, Xingyan Liu, Jiaqi Chen, Bang Zhang, Xin Zhou, Nan Du, and Jia Li. 2025. S^2R: Teaching LLMs to self-verify and self-correct via reinforcement learning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 22632–22654, Vienna, Austria. Association for Computational Linguistics.

Mathematical Association of America. 2024. 2024 american invitational mathematics examination. American Mathematics Competitions; problems archived by Art of Problem Solving.

Mathematical Association of America. 2025. 2025 american invitational mathematics examination. American Mathematics Competitions; problems archived by Art of Problem Solving.

Jingwei Ni, Ekaterina Fadeeva, Tianyi Wu, Mubashara Akhtar, Jiaheng Zhang, Elliott Ash, Markus Leippold, Timothy Baldwin, See-Kiong Ng, Artem Shelmanov, and Mrinmaya Sachan. 2026. Reprobe: Efficient test-time scaling of multi-step reasoning by probing internal states of large language models. *Preprint*, arXiv:2511.06209.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2023. GPQA: A graduate-level google-proof Q&A benchmark. *Preprint*, arXiv:2311.12022.

Rulin Shao, Shuyue Stella Li, Rui Xin, Scott Geng, Yiping Wang, Sewoong Oh, Simon Shaolei Du, Nathan Lambert, Sewon Min, Ranjay Krishna, Yulia Tsvetkov, Hannaneh Hajishirzi, Pang Wei Koh, and Luke Zettlemoyer. 2025. Spurious rewards: Rethinking training signals in RLVR. *Preprint*, arXiv:2506.10947.

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

Yiyang Shen, Lifu Tu, and Weiran Wang. 2026. Reinforcement learning-based knowledge distillation with llm-as-a-judge. *Preprint*, arXiv:2604.02621.

Peiyi Wang, Lei Li, Zhihong Shao, R. X. Xu, Damai Dai, Yifei Li, Deli Chen, Y. Wu, and Zhifang Sui. 2023a. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. *Preprint*, arXiv:2312.08935.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023b. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*.

Yibin Wang, Zhimin Li, Yuhang Zang, Chunyu Wang, Qinglin Lu, Cheng Jin, and Jiaqi Wang. 2025. Unified multimodal chain-of-thought reward model through reinforcement fine-tuning. *CoRR*, abs/2505.03318.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837.

Xiaomi LLM-Core Team, Bingquan Xia, Bowen Shen, Cici, Dawei Zhu, Di Zhang, Gang Wang, Hailin Zhang, Huaqiu Liu, Jiebao Xiao, Jinhao Dong, Liang Zhao, Peidian Li, Peng Wang, Shihua Yu, Shimao Chen, Weikun Wang, Wenhan Ma, Xiangwei Deng, and 4 others. 2025. Mimo: Unlocking the reasoning potential of language model – from pretraining to posttraining. *Preprint*, arXiv:2505.07608.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 42 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Qinan Yu, Alexa Tartaglini, Peter Hase, Carlos Guestrin, and Christopher Potts. 2026. Outcome rewards do not guarantee verifiable or causally important reasoning. *Preprint*, arXiv:2604.22074.

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. *Preprint*, arXiv:2503.14476.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Shiji Song, and Gao Huang. 2025. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *Preprint*, arXiv:2504.13837.

Liang Zeng, Liangjun Zhong, Liang Zhao, Tianwen Wei, Liu Yang, Jujie He, Cheng Cheng, Rui Hu, Yang Liu, Shuicheng Yan, Han Fang, and Yahui Zhou. 2024. Skywork-math: Data scaling laws for mathematical reasoning in large language models – the story goes on. *Preprint*, arXiv:2407.08348.

Wenlin Zhang, Xiangyang Li, Kuicai Dong, Yichao Wang, Pengyue Jia, Xiaopeng Li, Yingyi Zhang, Derong Xu, Zhaocheng Du, Huifeng Guo, Ruiming Tang, and Xiangyu Zhao. 2025. Process vs. outcome reward: Which is better for agentic RAG reinforcement learning. *Preprint*, arXiv:2505.14069.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Advances in Neural Information Processing Systems*, volume 36.

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. *Preprint*, arXiv:2311.07911.

Kun Zhou, Beichen Zhang, Jiapeng Wang, Zhipeng Chen, Wayne Xin Zhao, Jing Sha, Zhichao Sheng, Shijin Wang, and Ji-Rong Wen. 2024. Jiuzhang3.0: Efficiently improving mathematical reasoning by training small data synthesis models. *Preprint*, arXiv:2405.14365.

Lianghui Zhu, Xinggang Wang, and Xinlong Wang. 2025a. JudgeLM: Fine-tuned large language models are scalable judges. In *International Conference on Learning Representations*.

Xinyu Zhu, Mengzhou Xia, Zhepei Wei, Wei-Lin Chen, Danqi Chen, and Yu Meng. 2025b. The surprising effectiveness of negative reinforcement in llm reasoning. *Preprint*, arXiv:2506.01347.

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>

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 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

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