Efficient RLVR
(Data & Computation)

Hang Yang, Gio Song, Lisa Zhu

Act Only When It Pays: Efficient Reinforcement Learning for LLM Reasoning via Selective Rollouts

By Hang Yang

Background

RL Powers LLM Reasoning

Reasoning models leverage **Chain-of-Thought** (CoT) for stronger reasoning **(**e.g., **OpenAI o1, DeepSeek R1)**

Key driver: Reinforcement learning (RL) enable iterative strategy refinement with PPO and GRPO

Importantly, at **rollout** stage, generating more prompts can **further enhance training**

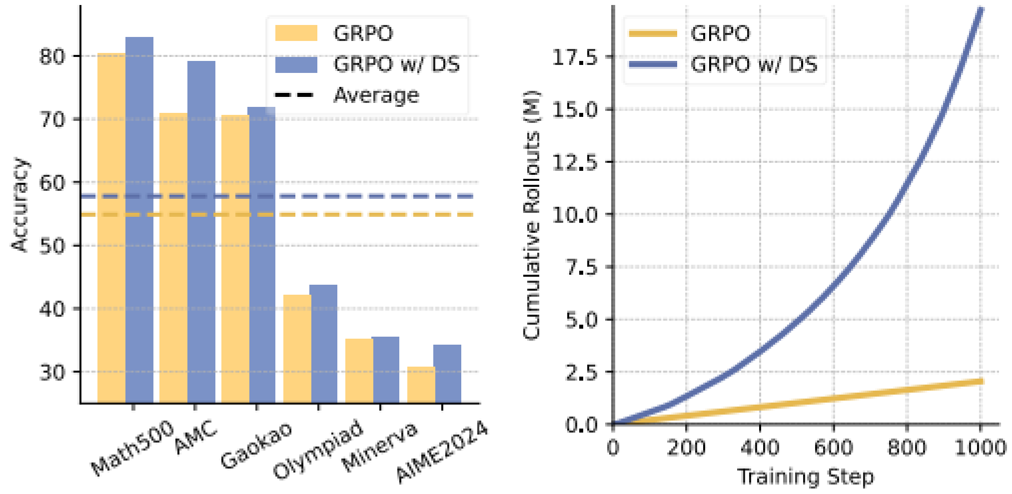
The main Challenge - Computational Resources

Rollout Scaling Benefits

higher-quality data

Stabilizes RL training

Improves model convergence



How to focus on sampling more valuable prompts?

Existing Methods

Static Data Pruning

- → A new model to identify valuable data point (No conclusive evidence of improving overall efficiency)
- → The value of a data point varies across models and stages (non-adaptive)

Dynamic Sampling

- → oversampling and filter out uninformative data only after rollout (additional rollout cost)

Ideal selective rollout algorithm →

- Online data selection
- Self-adaptive rollout strategy
- Low computational overheads

From the observation and analysis of GRPO to propose a new algorithm GRESO

Group Relative Policy Optimization (GRPO)

Objective function

$$\mathcal{J}_{GRPO}(\theta)=\mathbb{E}[q\sim P(Q),\{o_i\}_{i=1}^{G}\sim\pi_{\theta_{old}}(O|q)]$$
$$\frac{1}{G}\sum_{i=1}^{G}\left(\min\left(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{old}}(o_i|q)}A_i,\text{clip}\left(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{old}}(o_i|q)},1-\epsilon,1+\epsilon\right)A_i\right)-\beta\mathbb{D}_{KL}(\pi_\theta||\pi_{ref})\right)$$

$$A_{i,t}=\frac{r_i-\text{mean}(\{R_i\}_{i=1}^{G})}{\text{std}(\{R_i\}_{i=1}^{G})}.$$

one prompt → a group of response corresponding with a group of rewards {r1, r2, r3,…,rG}

Ai,t → advantage function to evaluate whether an example can provide learning signal

Prompts

Uninformative

Output1： 5

Output2： 5

…

OutputG: 5

informative

Output1： 1

Output2： 5

…

OutputG: 23

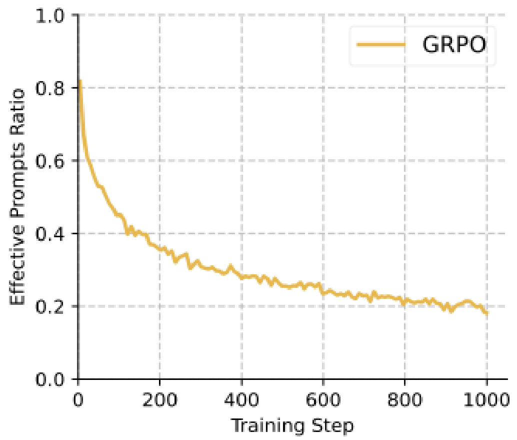
High Variance

Effective Prompts

GRPO Observations

Observation 1

Effective Prompts Ratio keeps decreasing as the training proceeds



Varying EPR hurt training stability and final model performance

Zero-Variance prompts

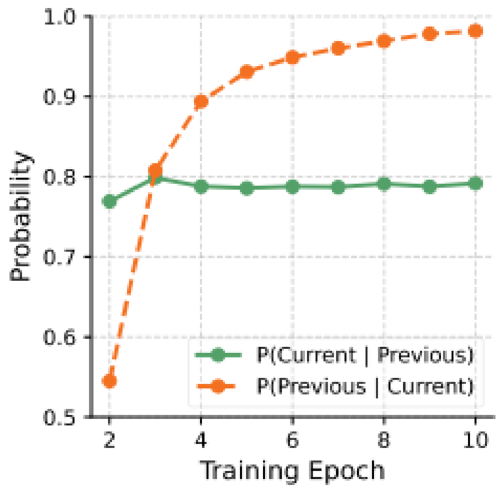
80%

5 times Rollouts

Maintain batch size

Identifying Priors to Rollout

Observation 2



The information value of a prompt is continuous and predictable over time

P(Previous | Current): 90% ~ P(Current | Previous) ~ 80%

in most cases it remains consistent (zero-variance stays zero-variance), but a small portion may transition.

Retain Potentially Valuable Prompts

Algorithm GRPO with Efficient Selective Rollout (GRESO)

Identifying **Priorly** ： formalize the problem of zero-variance prompt detection

$$T_i = (e_{i,1}, R_{i,1}), \dots, (e_{i,n}, R_{i,n}) \qquad R_{i,1} = \{r_{i,1}^{(k)}\}_{k=1}^{G}$$

$e_{i,j}$ denotes the epoch number （example x_i and j-th sampling）

$R_{i,1}$ represents the set of response rewards

To **predict** whether x_i is a zero-variance prompt

Probabilistic Pre-rollout Prompt Filtering :

$$p_f(x_i) = 1 - p_e^{z_i},$$

$$z_i = \max\left\{ k \in [0, n] \,\middle|\, \prod_{j=n-k+1}^{n} \mathbb{I}_{i,j} = 1 \right\},$$

$$\mathbb{I}_{i,j} = \begin{cases} 1, & \text{if all rewards in } R_{i,j} \text{ are identical}, \\ 0, & \text{otherwise}, \end{cases}$$

Algorithm GRPO with Efficient Selective Rollout (GRESO)

Probabilistic Pre-rollout Prompt Filtering :

$$p_f(x_i) = 1 - p_e^{z_i},$$

P_e denotes base exploration probability (P_e ↑ P_f ↓)

P_f denotes probability of Pre-rollout Prompt Filtering

$\mathcal{B} \leftarrow \emptyset$; $B_\mathrm{r} \leftarrow B_\mathrm{r}^{\mathrm{default}}$; $n_{easy}, n_{hard}, n_{total} \leftarrow 0,0,0$;
/* Rollout Stage.
repeat
    $\{x_i\}_{i=1}^{B_\mathrm{r}} \leftarrow$ Sample prompts from $\mathcal{D}$ and filter with Eq. 3 until batch size $= B_\mathrm{r}$;
    $\{x_i, r_i\}_{i=1}^{B_\mathrm{r} \times G} \leftarrow$ Rollout generation on $\{x_i\}_{i=1}^{B_\mathrm{r}}$;
    $\{x_i, r_i\}_{i=1}^{B_\mathrm{f} \times G} \leftarrow$ filter out zero-var prompt in $\{x_i, r_i\}_{i=1}^{B_\mathrm{r} \times G}$;
    $n_{\mathrm{easy}} \leftarrow n_{\mathrm{easy}} +$ filtered easy zero-var prompt count;
    $n_{\mathrm{hard}} \leftarrow n_{\mathrm{hard}} +$ filtered hard zero-var prompt count;
    $n_{\mathrm{total}} \leftarrow n_{\mathrm{total}} + B_\mathrm{r}$;
    $\mathcal{B} \leftarrow \mathcal{B} \bigcup \{x_i, r_i\}_{i=1}^{B_\mathrm{f} \times G}$;
    /* Adaptive rollout batch size.
    $B_\mathrm{r} \leftarrow \min(B_\mathrm{r}^{\mathrm{default}}$, Adaptive rollout batch size calculated by Eq. 6);
until $|\mathcal{B}| \geq B_t$;
/* Adjust Base Exploration Probability.
if $n_{easy}/n_{total} \geq \alpha_{easy}$ **then** $p_{easy} \leftarrow p_{easy} - \Delta p$;
else $p_{easy} \leftarrow p_{easy} + \Delta p$;
if $n_{hard}/n_{total} \geq \alpha_{hard}$ **then** $p_{hard} \leftarrow p_{hard} - \Delta p$;
else $p_{hard} \leftarrow p_{hard} + \Delta p$;
/* GRPO Training.
$\mathcal{B} \leftarrow$ select B_t examples from $\mathcal{B}$;
Update actor model with GRPO on $\mathcal{B}$;

Dynamically adjusting P_e and Batchsize

n_filtered /n_total > α → **P_e ↓**

More probability to filter zero-variance

$$B_r = \min\left(B_r^{default},\ \beta \frac{B_\Delta}{(1-\alpha)}\right)$$

Dynamical batchsize → no extra waste

Experiment

End-to-end Efficiency Comparison

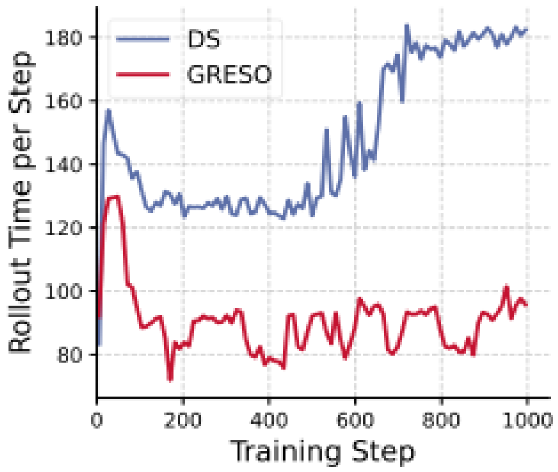
Dataset	Method	Math500	AIME24	AMC	Gaokao	Miner.	Olymp.	Avg.	# Rollout
Qwen2.5-Math-1.5B									
DM	DS	77.3	16.7	61.7	64.2	31.8	38.7	48.4	7.6M
	GRESO	76.6	15.0	61.4	66.2	33.3	38.5	48.5	**3.3M**
OR1	DS	77.1	16.7	50.3	65.5	30.9	39.7	46.7	3.8M
	GRESO	76.1	20.0	50.6	65.1	30.0	39.2	46.8	**1.6M**
DeepSeek-R1-Distill-Qwen-1.5B									
DM	DS	87.9	36.7	71.7	78.7	35.3	55.9	61.0	2.4M
	GRESO	87.7	36.7	71.1	78.4	33.9	55.1	60.5	**1.6M**
OR1	DS	84.8	25.0	68.4	74.0	34.1	54.2	56.7	2.4M
	GRESO	85.9	26.7	66.9	75.2	33.6	55.5	57.3	**1.5M**
Qwen2.5-Math-7B									
DM	DS	82.9	34.2	79.2	71.7	35.4	43.6	57.8	13.1M
	GRESO	82.2	32.5	80.7	70.2	35.4	44.1	57.5	**6.3M**
OR1	DS	82.9	34.2	63.1	67.3	34.9	46.3	54.8	11.4M
	GRESO	82.3	35.0	64.5	66.8	36.5	45.7	55.1	**3.4M**

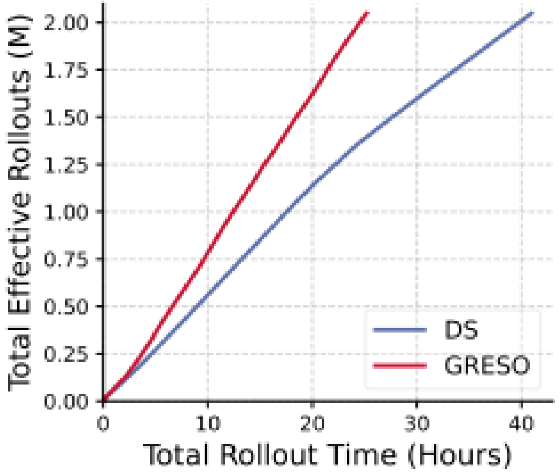
Method	Training	Other	Rollout	Total
Qwen2.5-Math-1.5B				
DS	8.1	3.6	41.0 (1.0×)	52.6 (1.0×)
GRESO	8.9	3.9	**25.2** (**1.6×**)	**37.9** (**1.4×**)
DeepSeek-R1-Distill-Qwen-1.5B				
DS	6.1	3.3	92.4 (1.0×)	101.9 (1.0×)
GRESO	6.8	4.0	**62.0** (**1.5×**)	**72.7** (**1.4×**)
Qwen2.5-Math-7B				
DS	16.1	6.1	155.9 (1.0×)	178.0 (1.0×)
GRESO	16.6	6.3	**65.5** (**2.4×**)	**88.3** (**2.0×**)

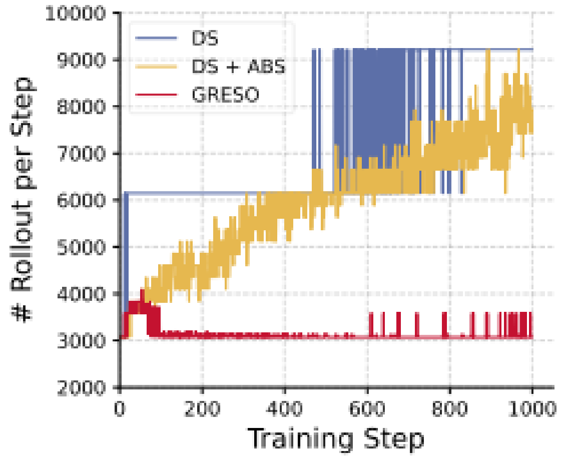
No performance drop with
up to 3.35× fewer rollouts and up to 2.4x wall-clock time speed-up

Experiment

Analysis and Ablation Study





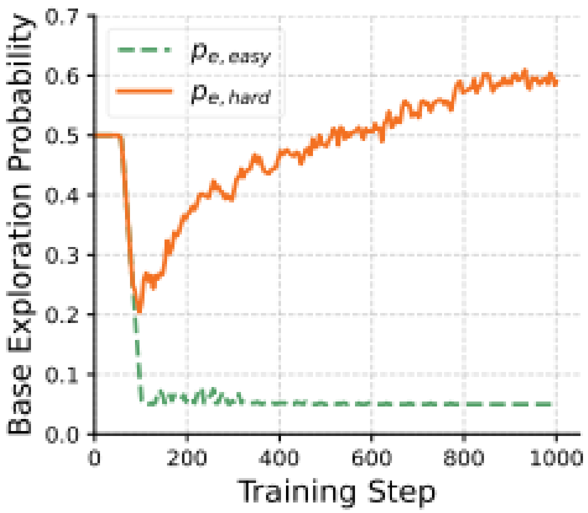


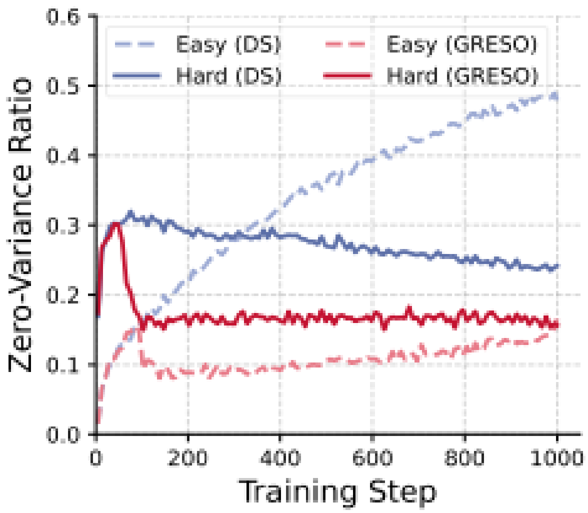
DS： Filters zero-variance prompts after rollout, but effective ratio drops and costs rise

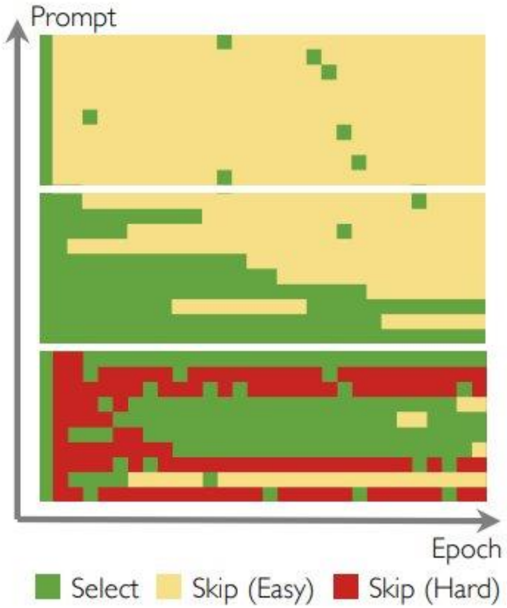
GRESO： Skips zero-variance prompts before rollout, keeping >70% effective ratio and lower cost

Experiment

Dynamics of self-adjustable base exploration probabilities.







GRESO adaptively adjusts exploration probabilities without manual tuning

As the model improves, Pe increases to explore harder examples

Conclusion

Key Contribution

GRESO : Act only when it pays, a novel algorithm to optimize rollout selection

3.35x fewer rollouts

2.4x rollout Speed up

2.0x overall training Speed up

Future Prospects

Extending selective rollouts to broader domains and more sophisticated data selection

Beyond the 80/20 Rule: High-Entropy Minority Tokens Drive Effective Reinforcement Learning for LLM Reasoning

By Gio Song

Background

Why Token-Level Analysis in RLVR Matters

- Reinforcement Learning for Verifiable Reasoning (RLVR) has become the **standard alignment method** for LLMs. But it shows only *moderate* gains
- Most prior work focuses on:
    - Algorithmic innovation (e.g., DAPO)
    - Task adaptation beyond math (e.g., Absolute Zero)
    - Empirical tricks (e.g., One-shot training)
- ❗ Missing: analysis of **how specific tokens** contribute to performance

Why This Paper?

So What Are We Missing in RLVR?

- Prior work treats all tokens equally during training
- But not all tokens are equally important in reasoning!
- Question: Can we identify and optimize the *right tokens*?

Quote for emphasis:

- “High-entropy tokens may decide reasoning *paths*, not just language *forms*.”
- Studying tokens, in fact, means studying the conditional probability distribution of the next token output by an LLM.

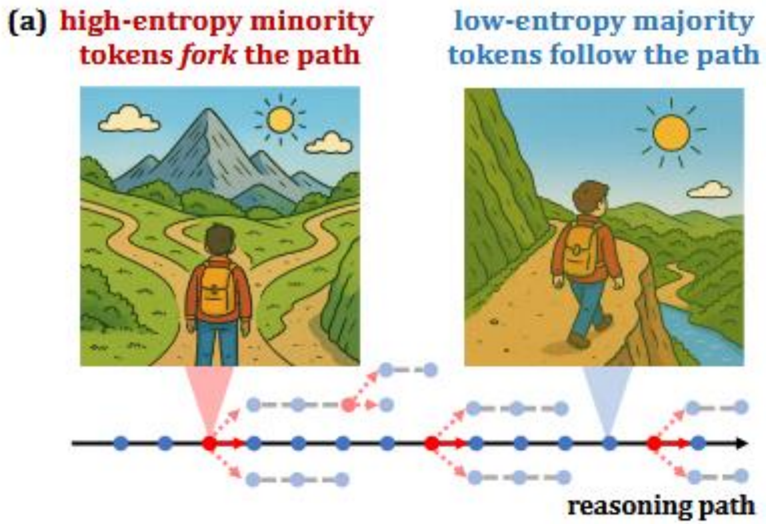
Key insights

Token Type	Entropy	Role in Output
Low-entropy	Very stable	Fills in predictable structure (e.g., math formulas, code)
High-entropy	Uncertain	Drives reasoning direction; controls "forks" in logic

Example:

In decimal, 1+1=2.But how does that translate to base 2?Well, in binary [..]

Blue tokens = low-entropy; red tokens = high-entropy (forking tokens)



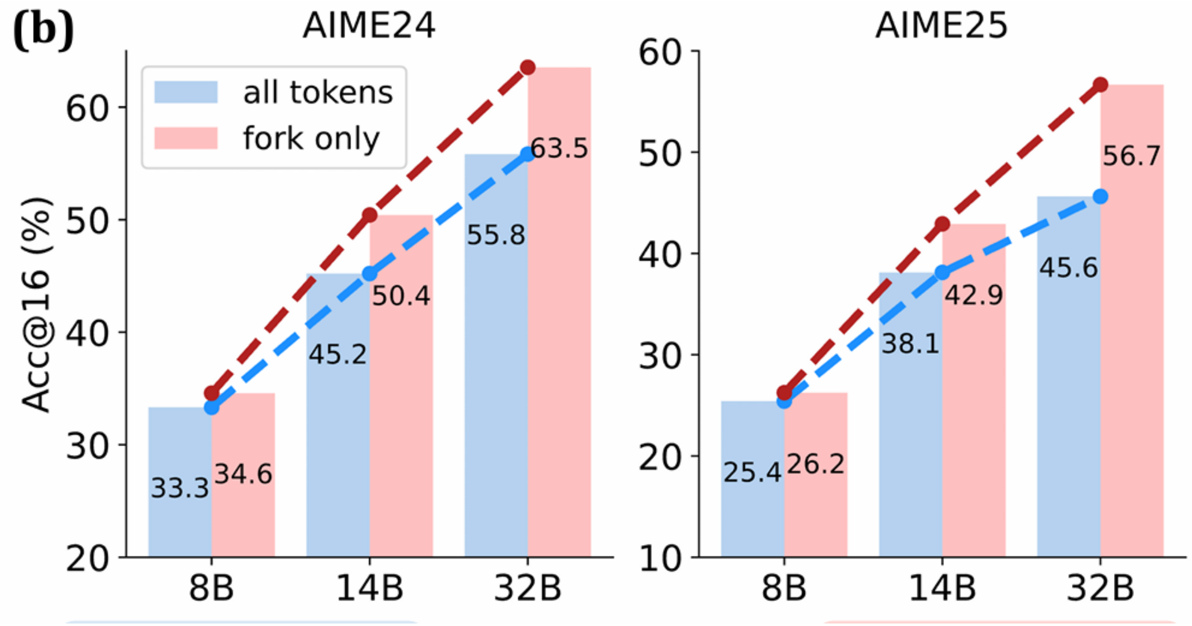
Further Discoveries

- **Slightly increasing entropy of high-entropy tokens** improves performance
- RLVR primarily **adjusts the entropy of high-entropy tokens**, while low-entropy tokens remain largely unchanged

Main Experiment & Ablation Experiment

Based on earlier findings, the authors hypothesize that:

- **Optimizing the conditional distributions of low-entropy tokens is unnecessary**.
- Instead, **only high-entropy tokens** (≈20% of all tokens) need targeted gradient updates to replicate most of the RL benefits.



The authors also **tune the proportion** of tokens to treat as "high-entropy" and find:

- **20% is optimal** for balancing performance and gradient efficiency.

Preliminaries

1.Token Entropy

Token entropy is based on the conditional probability distribution **over the vocabulary** at each step, not the specific token identity.

$$H_t = -\sum_{j=1}^{V} p_{t,j} \log p_{t,j}, \quad \text{where } p_t = \text{Softmax}\left(\frac{z_t}{T}\right)$$

2.DAPO – Dynamic sAmpling Policy Optimization

- DAPO selects **partially correct** prompts for training.
- Encourages learning from **useful but imperfect** trajectories.
- Advantage estimation ensures training focuses on **relatively better samples**.

$$\mathcal{J}_{\text{DAPO}}(\theta) = \mathbb{E}\left[\frac{1}{\sum_{i=1}^{G}|o^i|}\sum_{i=1}^{G}\sum_{t=1}^{|o^i|}\min\left(r_t^i(\theta)\hat{A}_t^i,\ \text{clip}(r_t^i(\theta), 1-\epsilon_{\text{low}}, 1+\epsilon_{\text{high}})\hat{A}_t^i\right)\right]$$

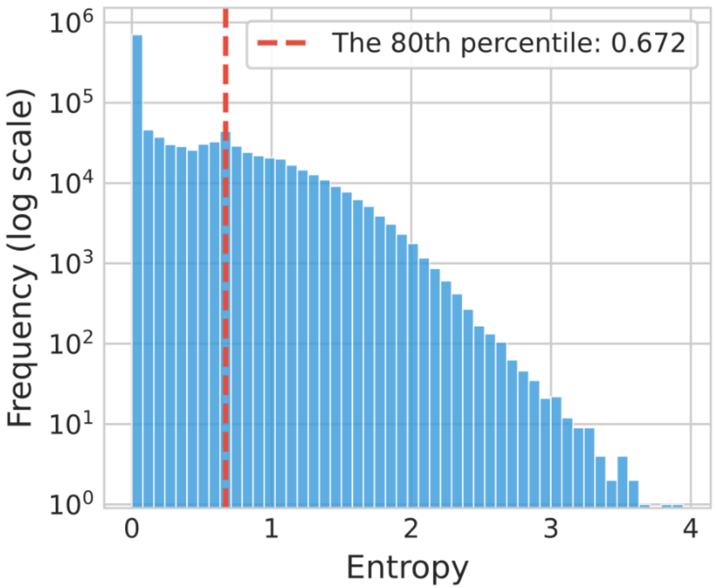
Pre--Experiment

3.1 Token Entropy in Chain-of-Thought (CoT)

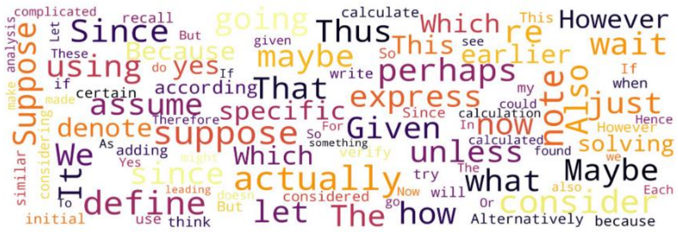
- **Goal**: Analyze entropy distributions in CoT outputs

Key Analysis:

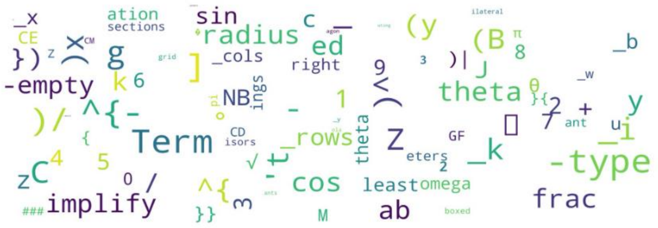
- **Token Entropy Distribution**:
  - Only **20% of tokens** have entropy > **0.672**
  - Most tokens are low-entropy — structural or formulaic
  - High-entropy tokens are rare, but impactful
- **Word Cloud Visualization**



(a) Distribution of token entropy



(b) Frequent tokens with the highest average entropy



(c) Frequent tokens with the lowest average entropy

Conclusion:

High-entropy tokens play a **decisive role in branching logic**
They are termed “forking tokens”

Entropy Intervention Experiment

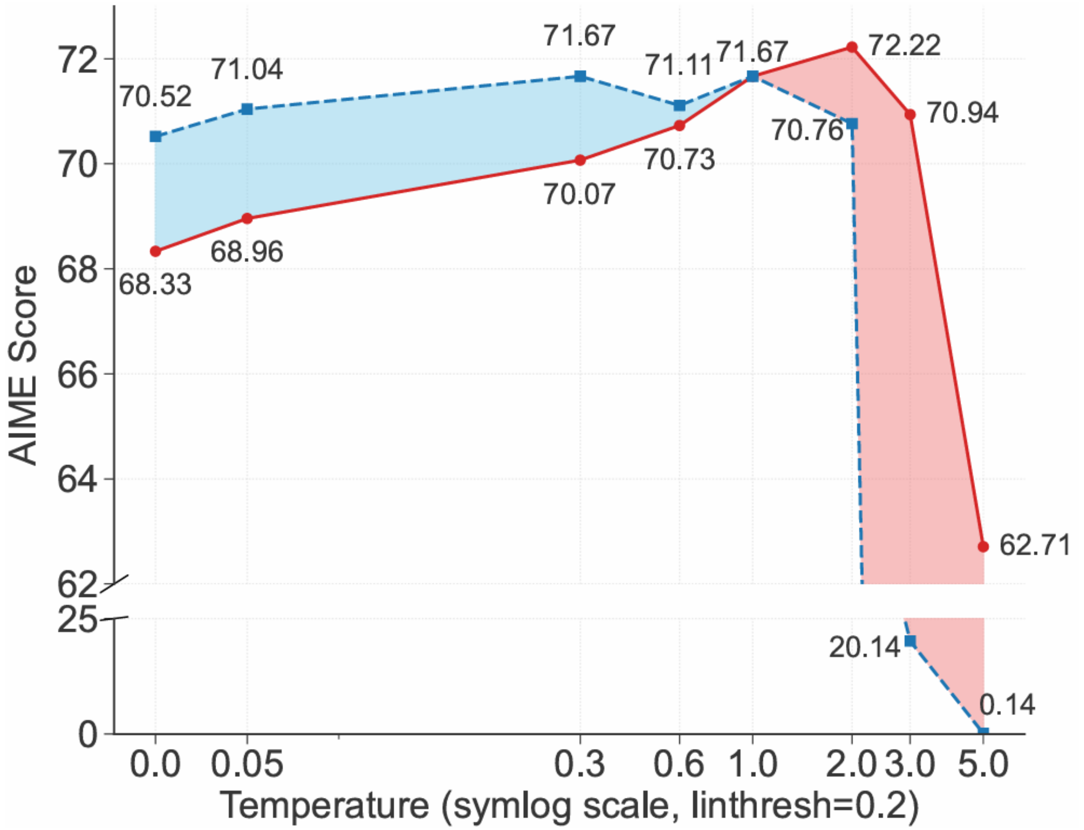


Figure 3: Average scores of AIME 2024 and AIME 2025. Red curve varying T_{high} with $T_{\text{low}} = 1$. Blue curve varying T_{low} with $T_{\text{high}} = 1$.

Method:

- Define threshold: $H_{threshold}$=0.672
- Use adaptive temperature scaling:

$$T'_t = \begin{cases} T_{\text{high}} & \text{if } H_t > H_{\text{threshold}} \\ T_{\text{low}} & \text{otherwise} \end{cases}$$

- Test two conditions:
  - Fix T_{low}=1, vary T_{high} (**Red Curve**)
  - Fix T_{high}=1, vary T_{low} (**Blue Curve**)

Insight:

Selectively **increasing entropy at forking tokens** improves Reasoning

This **mirrors the effect of RL training**, where entropy change is concentrated at decision-critical points

Pre--Experiment

3.2: RLVR Retains and Strengthens Entropy Patterns of Base Models

1) RLVR **Retains Entropy Structure of the Base Model**

Compare the **top 20% high-entropy tokens** between:

- **Base model**
- **Intermediate RLVR models**
- **Final RLVR model**

86% of high-entropy tokens remain consistent

Table 1: The progression of the overlap ratio in the positions of the top 20% high-entropy tokens, comparing the base model (i.e., step 0) with the model after RLVR training (i.e., step 1360).

Compared w/	Step 0	Step 16	Step 112	Step 160	Step 480	Step 800	Step 864	Step 840	Step 1280	Step 1360
Base Model	100%	98.92%	98.70%	93.04%	93.02%	93.03%	87.45%	87.22%	87.09%	86.67%
RLVR Model	86.67%	86.71%	86.83%	90.64%	90.65%	90.64%	96.61%	97.07%	97.34%	100%

Pre--Experiment

3.2: RLVR Retains and Strengthens Entropy Patterns of Base Models

2) RLVR **Selective Entropy Adjustment**:

- Tokens grouped by **5% entropy percentile intervals** (from low to high)
- Compute **average entropy change** after RLVR for each group

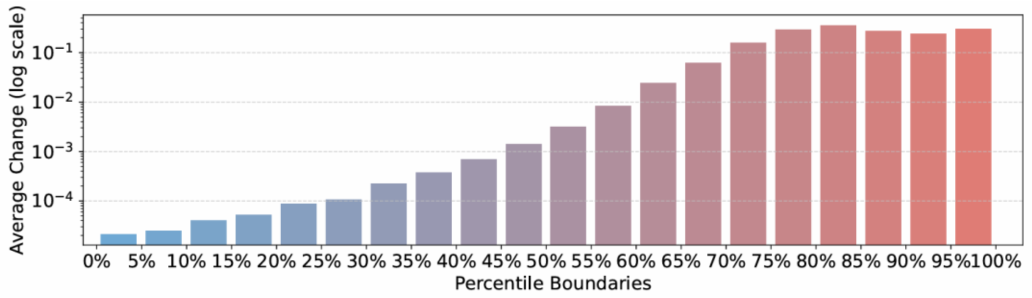


Figure 4: Average entropy change after RLVR within each 5% entropy percentile range of the base model. $x\%$ percentile means that $x\%$ of the tokens in the dataset have entropy values less than or equal to this value. It is worth noting that the Y-axis is presented on a *log scale*. Tokens with higher initial entropy tend to experience greater entropy increases after RLVR.

RLVR keeps the original token distribution structure intact
but **selectively increases entropy for a small set** (top 20%) of tokens
This sets the foundation for training **only high-entropy tokens** in later sections.

Main--Experiment

Adapted DAPO objective for only **high-entropy tokens**:

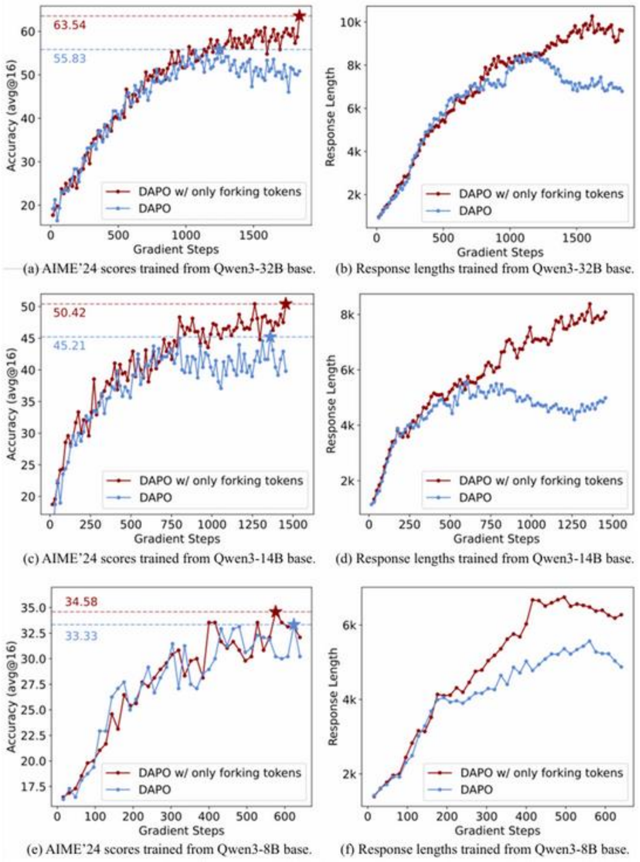
$$\mathcal{J}^{B}_{\text{HighEnt}}(\theta) = \mathbb{E}\left[\cdots \mathbb{I}(H^i_t \geq \tau^B_p)\min(r^i_t(\theta)\hat{A}^i_t, \text{clip}(\cdot))\right]$$

- Only tokens with entropy ≥ top-p threshold are used
- This means **RL updates only the most informative tokens**

Reinforcement learning performance boost is largely driven by forking tokens

Table 2: Comparison between *vanilla DAPO using all tokens* and *DAPO using only the top 20% high-entropy tokens (i.e. forking tokens)* in policy gradient loss, evaluated on the *Qwen3-32B*, *Qwen3-14B* and *Qwen3-8B* base models. "Acc@16" and "Len@16" denotes the average accuracy and response length over 16 evaluations per benchmark, respectively.

Benchmark	DAPO w/ All Tokens		DAPO w/ Forking Tokens		Improvement	
	Acc@16	Len@16	Acc@16	Len@16	Acc@16	Len@16
RLVR from the Qwen3-32B Base Model						
AIME'24	55.83	9644.15	**63.54**	12197.54	+7.71	+2553.39
AIME'25	45.63	9037.48	**56.67**	11842.25	+11.04	+2804.77
AMC'23	91.88	5285.03	**94.22**	5896.47	+2.34	+611.44
MATH500	94.36	2853.51	**94.88**	3366.01	+0.52	+512.5
Minerva	45.70	2675.28	**45.82**	2759.88	+0.12	+84.6
Olympiad	66.16	5597.37	**69.02**	7300.01	+2.86	+1702.64
Average	66.59	5848.80	**70.69**	7227.03	+4.10	+1378.22
RLVR from the Qwen3-14B Base Model						
AIME'24	45.21	7945.15	**50.42**	11814.36	+5.21	+3869.21
AIME'25	38.13	7056.98	**42.92**	12060.48	+4.79	+5003.5
AMC'23	89.53	4509.37	**91.56**	7095.13	+2.03	+2585.76
MATH500	92.23	2348.22	**93.59**	3970.10	+1.37	+1621.88
Minerva	42.16	2011.16	**43.20**	2959.32	+1.03	+948.16
Olympiad	61.14	4642.07	**64.62**	7871.25	+3.48	+3229.18
Average	61.40	4752.16	**64.39**	7628.44	+2.99	+2876.28
RLVR from the Qwen3-8B Base Model						
AIME'24	33.33	6884.89	**34.58**	9494.29	+1.25	+2609.40
AIME'25	25.42	5915.91	**26.25**	8120.20	+0.83	+2204.29
AMC'23	**77.81**	3967.91	77.19	5450.62	-0.625	+1482.71
MATH500	89.24	2059.00	**89.70**	2672.91	+0.46	+613.91
Minerva	39.77	1450.68	**40.26**	2068.41	+0.48	+617.73
Olympiad	56.67	3853.55	**57.43**	5241.54	+0.76	+1387.99
Average	53.71	4021.99	**54.23**	5508.00	+0.53	+1486.01



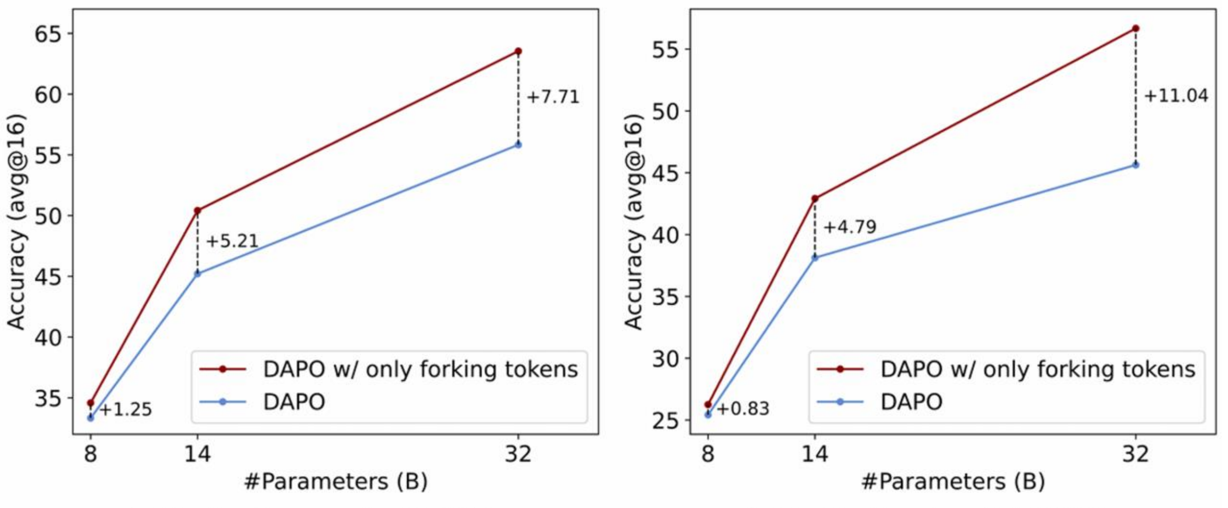
(a) AIME'24 scores trained from Qwen3-32B base. (b) Response lengths trained from Qwen3-32B base.

(c) AIME'24 scores trained from Qwen3-14B base. (d) Response lengths trained from Qwen3-14B base.

(e) AIME'24 scores trained from Qwen3-8B base. (f) Response lengths trained from Qwen3-8B base.

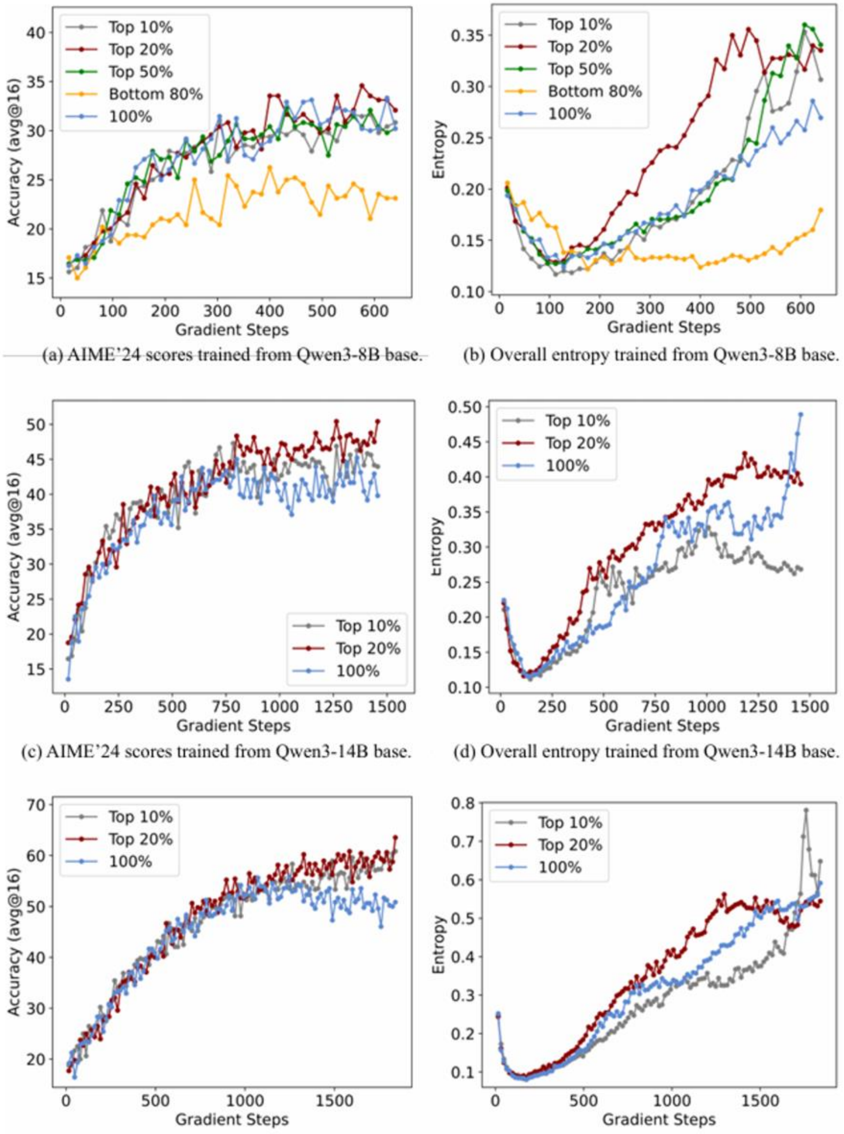
Further--Experiment

1. Varying ρ (proportion of high-entropy tokens)
2. Model Size Impact



(a) Scaling Trend on AIME 2024 (b) Scaling Trend on AIME 2025

Figure 8: Scaling trend of DAPO using only forking tokens (i.e., top 20% of high-entropy tokens) in policy gradient loss. These results suggest that concentrating exclusively on forking tokens in the policy gradient loss may yield greater benefits in larger reasoning models.



(a) AIME'24 scores trained from Qwen3-8B base. (b) Overall entropy trained from Qwen3-8B base.

(c) AIME'24 scores trained from Qwen3-14B base. (d) Overall entropy trained from Qwen3-14B base.

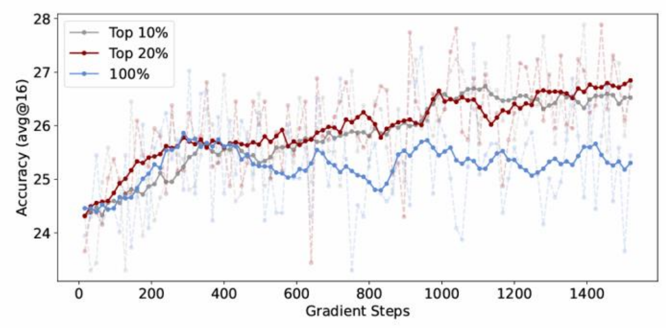
(e) AIME'24 scores trained from Qwen3-32B base. (f) Overall entropy trained from Qwen3-32B base.

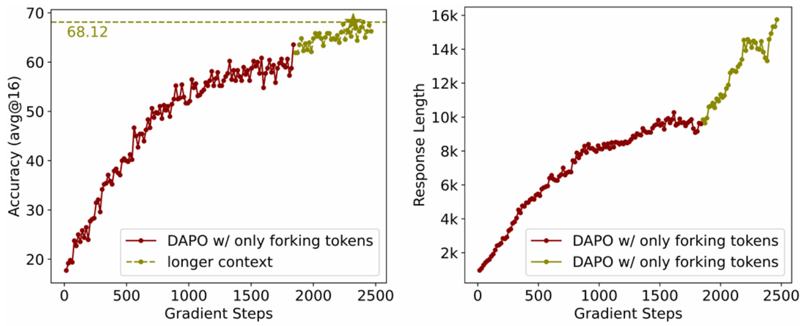
Smaller subset of tokens (high entropy) can drive **stronger performance**, reducing cost while increasing quality.

--foundational claim of the article

Analysis

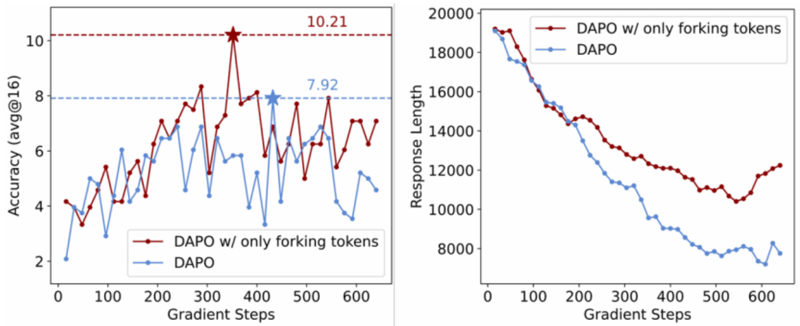
Aspect	Finding
Cross-task generalization	High-entropy token updates **improve transfer** (math → code)
Long-context reasoning	Training with forking tokens supports longer outputs and deeper logic
Portability to smaller models	Works well even under **low-compute, small-model cold-start** scenarios. model-agnostic





(a) AIME'24 scores

(b) Response length



(a) AIME'24 scores trained from Llama-3.1-8B after cold-start.

(b) Response length trained from Llama-3.1-8B after cold-start.

Discussion, Conclusion & Limitations

Discussion & Conclusions

- Why High-Entropy Tokens Matter in RL
- LLM CoT and Token Entropy
- Why RLVR Works

Limitations & Further Improvement

- Mainly on **Qwen models**.
- Dataset limited to mathematical reasoning.
- Results are experiment-specific.

Develop **better RLVR algorithms**

- Supervised fine-tuning (SFT)
- Distillation
- Inference pipelines
- Multi-modal training

Spurious Rewards: Rethinking Training Signals in RLVR

Lisa Zhu, Hang Yang, Gio Song

Core Idea & Findings

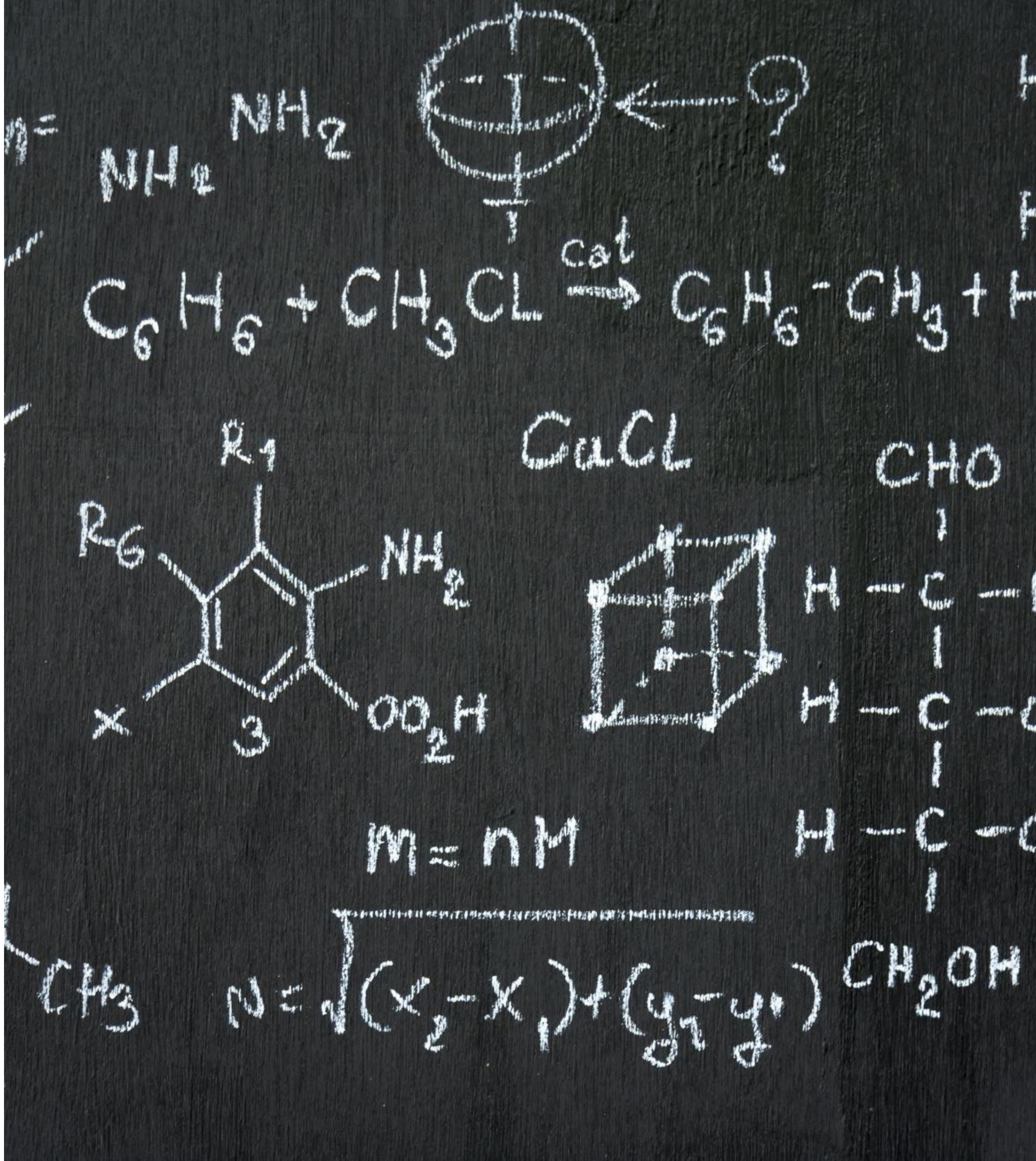
- Reinforcement Learning with Verifiable Rewards (RLVR) improves reasoning in LLMs
- Surprisingly, it works even with spurious rewards
  - Random, wrong, or irrelevant
- Qwen2.5-Math-7B
  - Random rewards: 21.4%
  - Wrong label: +24.1%
- Performance gains nearly match ground truth training

Additional Insights

- Model differences
  - Strong gains for Qwen2.5-Math
  - Little or negative effect on Llama3 & OLMo2
- Code reasoning (thinking in code without actual code execution):
  - Distinctive behavior for Qwen2.5-Math
  - Becomes more frequent after RLVR
  - From 65% → 90%
- Implication
  - RLVR surfacing latent abilities from pretraining
    - Not reward signal itself

Experiment & Results I

- Goal: Test if RLVR still improves reasoning with weaker or spurious rewards instead of ground truth
- Method:
  - Base model: Qwen2.5-Math
  - Training: GRPO algorithm, DeepScaleR dataset
    - GRPO finetune base model
    - DeepScaleR trained with spurious binary (0-1) reward functions
- Investigate the limits of how little supervision is needed for RLVR training



Experiment & Results II

- Types of rewards tested
  - Standard to Weak to Spurious
  - Ground Truth → Majority Vote → Format → Random →Incorrect
    - Ground Truth: Correct answers only
    - Incorrect: Deliberately reward wrong answers from pseudo-labeling

- Results
  - All reward types have significant math gains within 1st 50 steps
  - Smaller model also improves, but more slowly
  - Takeaway: RLVR boosts performance even with spurious signals

    → Elicit **latent abilities** from pretraining

Cross-Model Analysis

- Goal: Test if spurious-reward gains generalize across models
- Models: Qwen2.5-Math vs. OLMo2 vs. Llama3
- Findings:
  - Qwen2.5-Math: large gains even with spurious rewards
  - OLMo2 & Llama3: minimal or negative gains
- Why Qwen
  - Exhibits strong code reasoning (i.e. writes math steps in Python)
  - Accuracy: 61% with code vs. 28% without
- Takeaway: Spurious rewards amplify latent code-reasoning abilities in Qwen, not transferable to other model

What makes Qwen so SPECIAL?--Code Reasoning

- **Observation**: The Qwen2.5-Math model frequently generates Python code as a method of reasoning
- **More structured and accurate solutions**.
- **Evidence**: **65%+ code reasoning frequency**. After RLVR (even with random or incorrect rewards), this frequency rises above **90%**.
- **Why this matters**:
  Code reasoning leads to significantly **higher accuracy（60.9%）**. Only **35.0%** on responses with natural language reasoning.
- **Other models (e.g., OLMo, LLaMA)** either do not use code (No-Code) or use it ineffectively (Bad-Code), and hence don’t benefit similarly from RLVR.

MATH Question:

What is the distance, in units, between the points $(2, -6)$ and $(-4, 3)$? Express your answer in simplest radical form.

Qwen2.5-Math-7B Solution (correct):

To find the distance between two points (x_1, y_1) and (x_2, y_2) in a Cartesian plane...
Let’s break this down step-by-step and compute the result using Python.

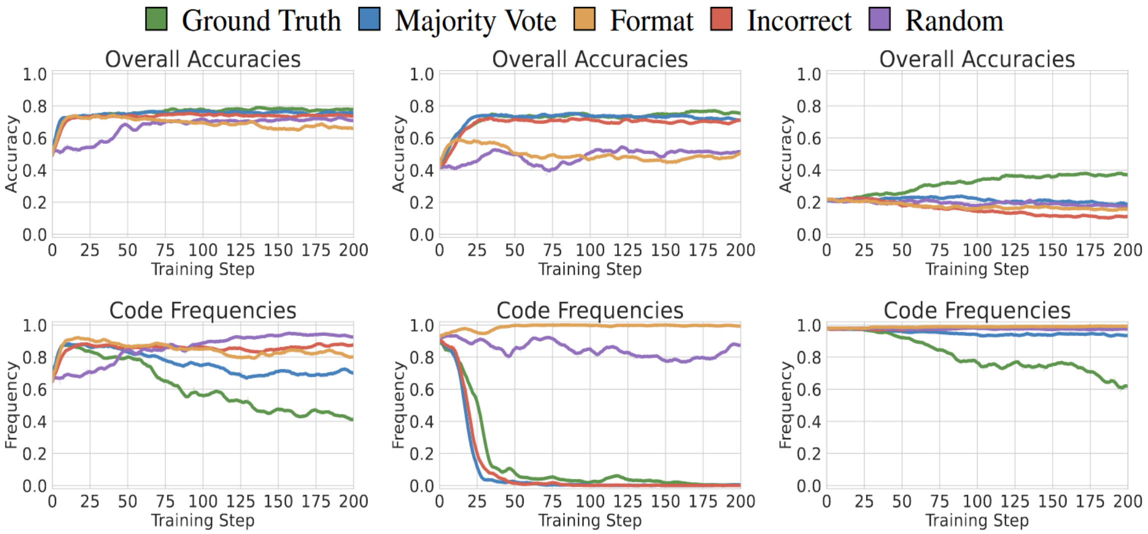
```
import math
...
# Calculate the distance using the distance formula
distance = math.sqrt(dx**2 + dy**2)
print(distance)
```

output: 10.816653826391969
...
Thus, the final answer is: $\boxed{3\sqrt{13}}$

Model	Qwen2.5-Math-7B	Qwen2.5-Math-1.5B	Qwen2.5-7B	OLMo2-7B-SFT
Code Frequency	65.0	53.6	92.2	98.0
Acc. w/ Code	60.9	52.6	39.9	21.0
Acc. w/ Lang	35.0	17.2	61.5	40.0

RLVR with Spurious Rewards Amplifies Pretrained Reasoning Strategies

- Why do spurious rewards work?
- **Evidence:** Code Reasoning Frequency Strongly Correlates with Accuracy
- **Before RLVR**: Qwen2.5-Math-7B uses code reasoning in 65% of outputs.
- **After RLVR**: rises to **90–95%**, and accuracy **increases alongside**.
- **Random reward** leads to slower increase but eventually hits **95.6%** code reasoning rate.
- **True label reward** causes an initial spike in code usage, but this later **declines** as the model learns to solve more via natural language.



(a) Qwen2.5-Math-7B (b) Qwen2.5-7B (c) OLMo2-7B-SFT

RLVR with Spurious Rewards Amplifies Pretrained Reasoning Strategies

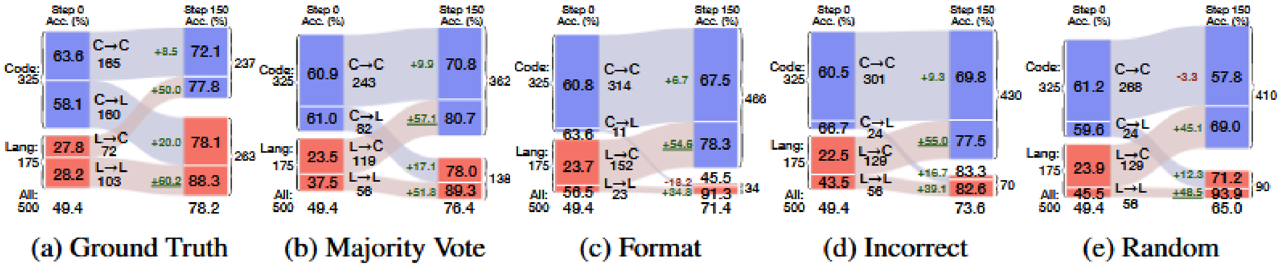
The authors examine performance shifts across 4 reasoning transition patterns:

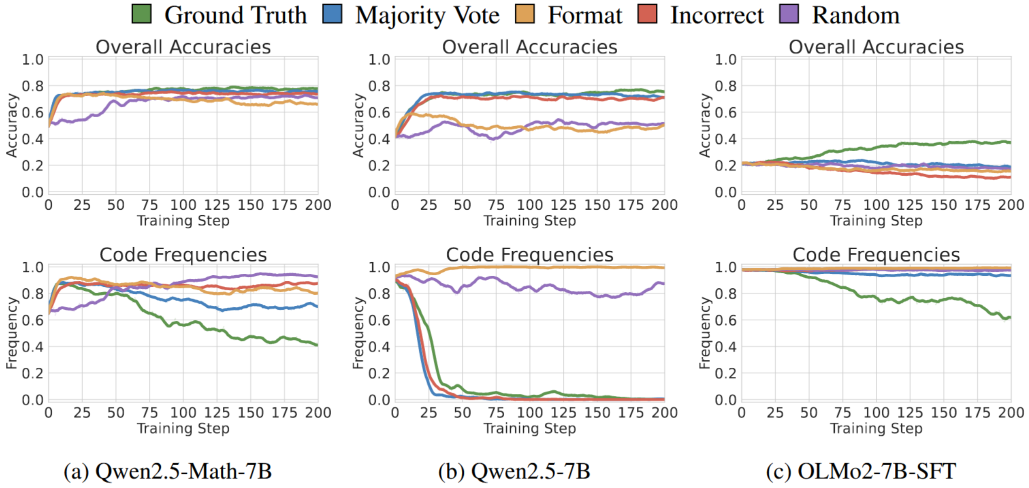
Code→Code	Code reasoning before and after training
Code→Lang	Switch from code to language reasoning
Lang→Code	Switch from language to code reasoning
Lang→Lang	Natural language reasoning both before and after

Two main metrics were tracked:

- Subset **frequency** (how often that strategy occurred)
- Subset **accuracy** (how correct it was)

RLVR with Spurious Rewards Amplifies Pretrained Reasoning Strategies





Findings from Strategy Shift Analysis:

- Under **spurious and weak rewards**, Qwen2.5-Math-7B tends to:
  - Maintain code reasoning if it already used it. （Code→Lang）
  - **Switch from language to code reasoning** (Lang→Code) in most other cases.
- **True reward** does not cause the same shift

Other models behave differently:

- **Qwen2.5-7B** sees a **decline in code reasoning** under correct/majority/incorrect rewards
- **OLMo2-7B-SFT** also shows **decreased code use** under valid reward signals.
- **LLaMA and other No-Code models** show no meaningful change in strategy.

Analysis

Table 2: Partial contribution to the overall performance gain averaged over rewards that successfully steered the model's reasoning strategy (Figure 6).

Model	Qwen2.5-Math-7B	Qwen2.5-Math-1.5B	Qwen2.5-7B
Avg. Total Gain	↑ 23.5%	↑ 28.5%	↑ 30.6%
$C_{Code \to Code}$	11.6%	2.8%	0.2%
$C_{Code \to Lang}$	8.6%	2.0%	**93.9%**
$C_{Lang \to Code}$	**58.3%**	**78.7%**	0.0%
$C_{Lang \to Lang}$	21.4%	16.5%	5.9%

- **Qwen-Math models improve by switching into their strength (code reasoning)**.
- **Other models improve by abandoning inefficient strategies**, like code reasoning, in favor of simpler text reasoning.
- For Qwen2.5-Math, the performance gains from spurious reward **do not reflect new skill acquisition**, but rather **the amplification of a previously learned, effective strategy** (code reasoning).

- **RLVR, particularly with non-informative or even misleading reward signals, can still work extremely well — if and only if the underlying model has already internalized useful reasoning strategies during pretraining.**

Interventions on code reasoning

Impact of Increased Code Reasoning on Performance

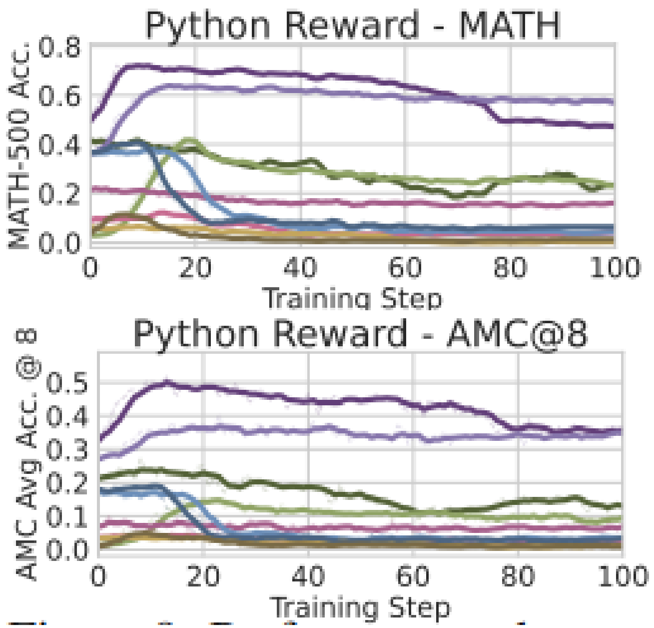
（1） Prompting (Answer begin with “let‘s solve this using python”)

Model	Original	Prompting	Abs. Diff.
Qwen2.5-Math-1.5B	36.2%	60.4%	+24.2%
Qwen2.5-Math-7B	49.4%	64.4%	+15.0%
Qwen2.5-1.5B	3.0%	13.0%	+10.0%
Qwen2.5-7B	41.6%	22.2%	–19.4%
Llama3.2-3B-Instruct	36.8%	8.2%	–28.6%
Llama3.1-8B-Instruct	36.8%	15.2%	–21.6%
OLMo2-7B	9.0%	7.8%	–1.2%
OLMo2-7B-SFT	21.4%	18.6%	–2.8%

Qwen-Math-7B Qwen-Math-1.5B Qwen-7B Qwen-1.5B
Olmo2-7B-SFT Olmo2-7B Llama3.1-8B
Llama3.2-3B Llama3.1-8B-Instruct Llama3.2-3B-Instruct

Qwen model : ⬆ Llama, OLMo : ⬇

（2） RLVR(Assign a positive rewards only answer contain “python”)

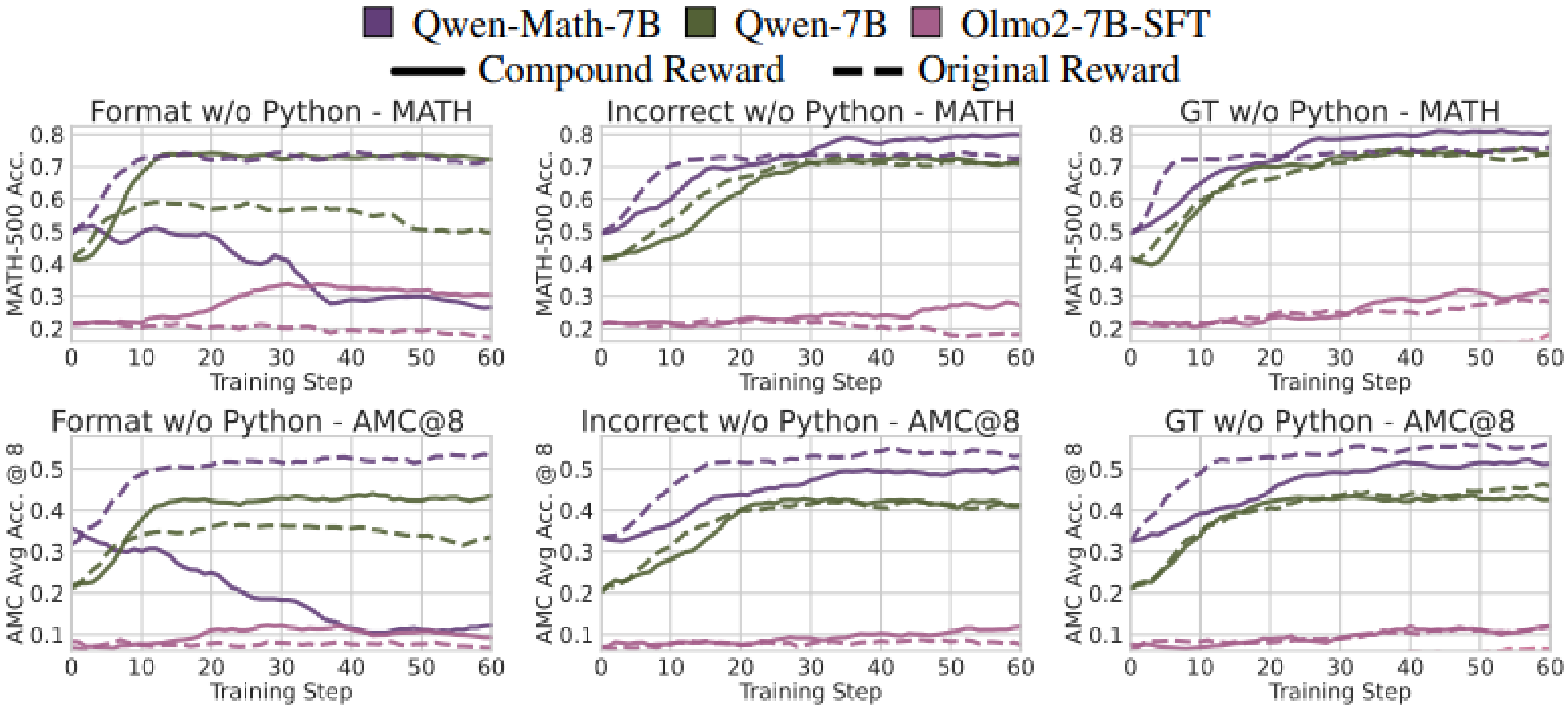


Qwen2.5-Math-7B model generated code reasoning in its' answer >99% just 20 training steps

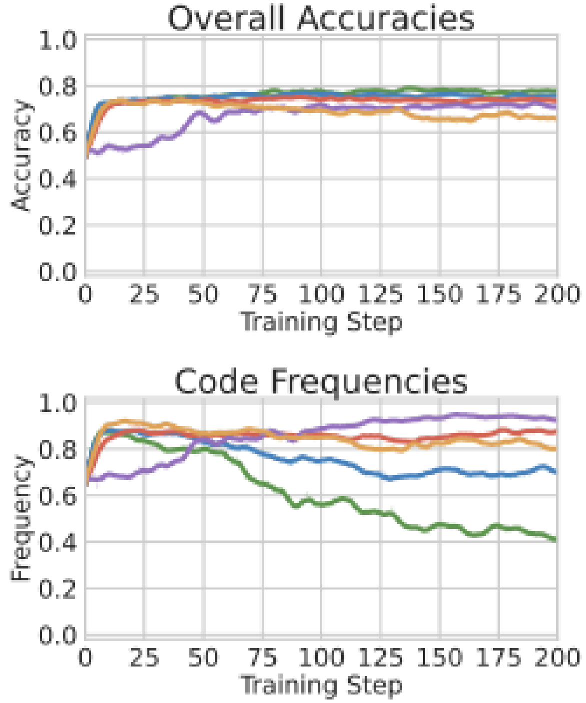
Inhibiting code reasoning during RLVR with spurious rewards

Reward a response if and only if:

(1) spurious reward condition (original) (2) no string "python" (compound)



(a) Format w/o Python (b) Incorrect w/o Python (c) Ground Truth w/o Python



(a) Qwen2.5-Math-7B

Qwen math model : (1) format reward↓ (2) Incorrect reward (AMC↓)

(3) Ground truth Performance improvement ≠ sole code reasoning frequency

Bad code model : Compoud rewards > Original (downweight suboptimal model behavior)

Curious cases： Training Signals from Incorrect Rewards and Random Rewards

Hypothesis： Incorrect Rewards → Reasoning

（1）many incorrect labels remain close to ground truth values （ positive reinforcement）

（2）incorrect labels may function like format reward （some degree of correct ）

Random Rewards → Reasoning

Hypothesis from someone ：most rewarded answers are correct （X）

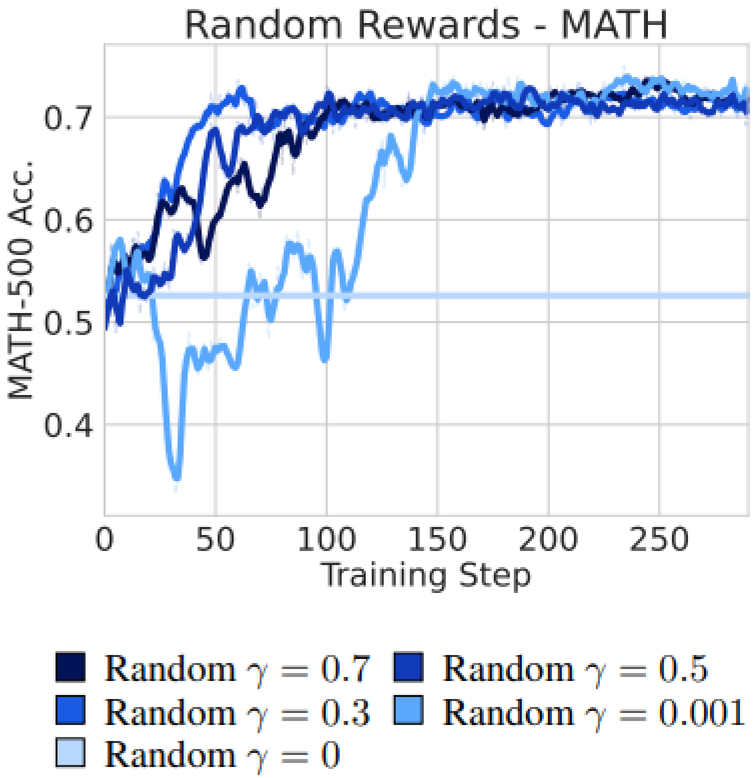
Rewarded response : correct > incorrect → Penalized response : correct > incorrect

Normalization of reward in GRPO → Random rewards ≠ bias toward correct answers

Why random rewards worked？

Why random rewards worked?

Experiment 1 ：
Random rewards with varying probabilities

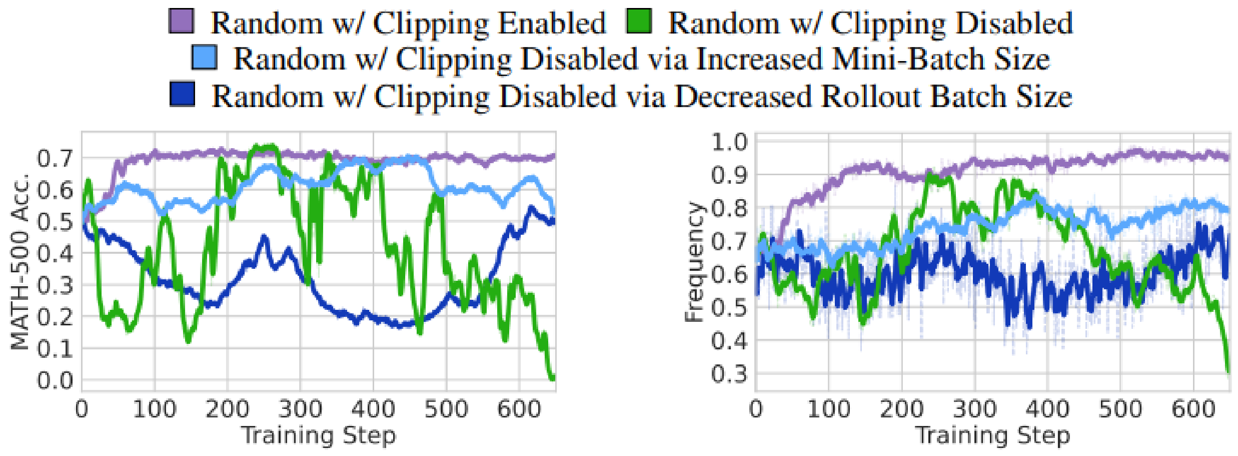


Except for γ = 0,
γ do not affect the final performance

Experiment 2：
Clipping function enabled Vs disabled

$$\mathcal{J}_{GRPO}(\theta)=\mathbb{E}[q\sim P(Q),\{o_i\}_{i=1}^{G}\sim\pi_{\theta_{old}}(O|q)]$$

$$\frac{1}{G}\sum_{i=1}^{G}\left(\min\left(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{old}}(o_i|q)}A_i,\text{clip}\left(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{old}}(o_i|q)},1-\epsilon,1+\epsilon\right)A_i\right)-\beta\mathbb{D}_{KL}(\pi_\theta||\pi_{ref})\right)$$



(a) Performance on MATH-500 (b) Frequency of Code Reasoning

（1） directly turning off the clipping term

（2） adjusting training and rollout batch sizes

（ πθ = πold） Clipping: ~21% performance gain

Optimizing algorithm's bias toward exploiting priors learned during pretraining (Amplify penalties, Regulate rewards)

Conclusion

Summary

（1） RLVR with spurious rewards (random, incorrect, format-only) improves Qwen2.5-Math by amplifying pre-existing code reasoning patterns rather than teaching new skills.

（2） Code reasoning frequency increases from 65% to 90%+ during training, directly correlating with performance gains across all reward types.

（3） Model-dependent effects — spurious rewards work for Qwen families but consistently fail for Llama and OLMo models

Key Implications

（1） Pretraining determines outcomes — RLVR effectiveness depends on what reasoning patterns already exist in the base model.

（2） Spurious signals can work — when they trigger beneficial pre-trained behaviors like code reasoning capabilities.

R-Zero: Self-Evolving LLM from Zero Data

By Lisa Zhu

Motivation

- LLMs need huge amounts of **human-curated data and labels** for fine-tuning
- **Costly**, **slow**, and **limits scalability** toward true self-evolving AI
- Existing “label-free” methods still rely on **pre-existing tasks** or **external verification**
- **R-Zero**: Fully **autonomous** framework
  - LLMs generates it **own training data** from scratch

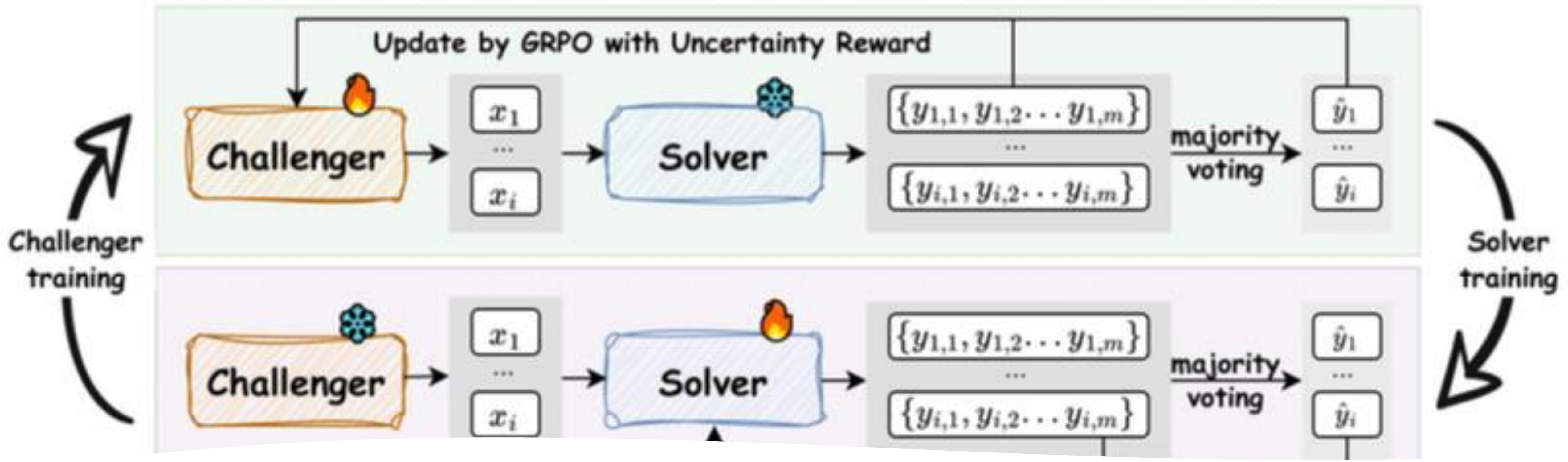
Preliminaries

Group Relative Policy Optimization(GRPO)

- **Reinforcement Learning algorithm** for fine-turning LLMs
- ~~Separate value function~~ Compares responses within the same group
- Uses **z-score normalization** of rewards: each answer is judged relative to others
- Encourages better responses while preventing large policy drift

Reinforcement Learning with Verifiable Rewards (RLVR)

- Paradigm for fine-tuning models
- Applies when response quality can be objectively checked
- Uses rule-based verifier
  - Reward = 1 if correct, 0 if wrong
- Foundation for training the Solver in R-Zero



Methodology Overview

- **R-Zero = Challenger + Solver, initialized from the same LLM.**
- **Works in an iterative loop:**
  - Challenger generates synthetic questions via GRPO.
  - Solver trains on these questions with pseudo-labels.
- **Self-supervised**: no human labels required.
- Goal: Challenger and Solver **co-evolve**, making Solver increasingly stronger

Challenger & Solver Training

Challenger (Qθ)

- Generates **challenging questions** via GRPO.
- Guided by reward signals (uncertainty, penalties).
- Goal: push Solver to face progressively harder tasks

Solver (Sϕ)

- Fine-tuned on Challenger's filtered question set.
- Uses GRPO with a **verifiable reward:**

$$r_j = \begin{cases} 1, & \text{if } x_j \text{ is identical to the pseudo-label } \tilde{y}_i, \\ 0, & \text{otherwise.} \end{cases}$$

- Learns to correctly answer increasingly difficult questions

$$r_{\text{uncertainty}}(x;\phi) = 1 - 2\left|\hat{p}(x;S_\phi) - \frac{1}{2}\right|$$

Reward Function – Uncertainty Reward

- Encourages **questions with mid-level difficulty**.
- Solver's accuracy on question x:
  $$\hat{p}(x;S_\phi) = \frac{1}{m}\sum_{j=1}^{m} \mathbb{1}\{y_j = \tilde{y}(x)\}$$
- Maximized when Solver accuracy ≈ **50%**, forcing learning on "frontier" problems

Repetition & Format Penalties

- **Repetition Penalty**
  - Prevents generating near-duplicate questions.
  - Uses **BLEU score similarity**; larger clusters → larger penalty.
  - Formula:

$$r_{\text{rep}}(x_i) = \lambda \frac{|C_k|}{B}$$

- **Format Check Penalty**
  - Structural rule: question must be enclosed in <question> & </question>
  - If not, reward = 0 and question is discarded

Reward Function – Composite Reward

- Purpose: Combine signals from uncertainty and repetition to train Challenger effectively.
- Formula:

$$r_i = \max(0, r_{\text{uncertainty}}(x_i; \phi) - r_{\text{rep}}(x_i))$$

- **Interpretation**:
  - Starts from **uncertainty reward** (challenging but solvable questions).
  - Subtracts penalty if question is too similar to others.
  - Ensures reward ≥ 0, preventing negative reinforcement.
- **Takeaway**: Final reward signal balances *difficulty* with *diversity*

Experiments Setup – Models & Training

- Models Tested
  - **Qwen3-4B / 8B** → scale within same family
  - **OctoThinker-3B / 8B** → different lineage (Llama-based)
  - Ensures evaluation across **two distinct architectures**
- Training Details
  - Candidate pool: **8,000 questions** per iteration
  - Solver samples 10 answers per question
  - Keep only mid-consistency tasks (**3–7 matched answers**)
  - **Rewards:** uncertainty (Solver confusion)

Experiments Setup – Benchmarks

- Mathematical Reasoning
  - 7 Benchmarks: AMC, Minerva, MATH-500, GSM8K, OlympiadBench, AIME-2024, AIME-2025
  - Test correctness, complexity, and comprehensiveness
  - Metrics reported:
    - AMC & AIME: mean@123
    - Others: accuracy (greedy decoding)
- General Domain Reasoning
  - **MMLU-Pro**: Harder multi-task questions (language model capabilities)
  - **SuperGPQA**: Graduate-level reasoning across 285 disciplines
  - **BBEH**: More difficult BIG-Bench tasks for complex reasoning

Math Reasoning Results

Scores improve with each iteration; first iteration already gives a strong boost, showing RL-trained Challenger is critical

- Findings
  - **Consistent gains across all models** (Qwen3 & OctoThinker families)
  - **Qwen3-8B:** +5.51 points (49.18 → 54.69 after 3 iterations)
  - **OctoThinker-3B:** +2.68 points (26.64 → 29.32)
  - Larger models improve more, but smaller ones still benefit
- Takeaway: R-Zero is **effective & model-agnostic**, boosting performance across scales and architectures

Model Name	AVG	AMC	Minerva	MATH	GSM8K	Olympiad	AIME25	AIME24
Qwen3-4B-Base								
Base Model	42.58	45.70	38.24	68.20	87.79	41.04	6.15	10.94
Base Challenger	44.36	45.00	45.22	72.80	87.87	41.19	7.29	11.15
R-Zero (Iter 1)	48.06	51.56	51.47	78.60	91.28	43.85	**9.17**	10.52
R-Zero (Iter 2)	48.44	52.50	51.47	**79.80**	91.66	44.30	4.27	**15.10**
R-Zero (Iter 3)	**49.07**	**57.27**	**52.94**	79.60	**92.12**	**44.59**	4.27	12.71
Qwen3-8B-Base								
Base Model	49.18	51.95	50.00	78.00	89.08	44.74	16.67	13.85
Base Challenger	51.87	60.70	57.72	81.60	92.56	46.44	13.44	10.62
R-Zero (Iter 1)	53.39	61.56	59.93	**82.00**	93.71	48.00	14.17	14.37
R-Zero (Iter 2)	53.84	61.56	59.93	**82.00**	93.93	48.30	17.60	13.54
R-Zero (Iter 3)	**54.69**	**61.67**	**60.66**	**82.00**	**94.09**	**48.89**	**19.17**	**16.35**
OctoThinker-3B								
Base Model	26.64	17.19	24.26	**55.00**	73.69	16.15	0.21	0.00
Base Challenger	27.51	20.19	24.63	54.60	74.98	15.70	0.10	**2.40**
R-Zero (Iter 1)	27.76	20.39	25.74	54.60	**75.51**	16.30	0.10	1.67
R-Zero (Iter 2)	28.20	24.06	25.37	54.80	74.45	17.48	0.00	1.25
R-Zero (Iter 3)	**29.32**	**27.03**	**27.57**	54.20	74.98	**18.22**	**3.23**	0.00
OctoThinker-8B								
Base Model	36.41	32.11	41.91	65.20	86.96	26.52	1.56	0.62
Base Challenger	36.98	29.30	42.28	66.20	**88.10**	27.56	1.04	4.38
R-Zero (Iter 1)	37.80	32.97	45.22	65.60	86.96	**28.44**	**1.98**	3.44
R-Zero (Iter 2)	38.23	32.58	**48.53**	67.20	87.11	27.26	0.00	**4.90**
R-Zero (Iter 3)	**38.52**	**34.03**	48.22	**68.80**	87.19	27.56	0.42	3.44

General Results Reasoning

Model Name	Overall AVG	MATH AVG	SuperGPQA	MMLU-Pro	BBEH
Qwen3-4B-Base					
Base Model	27.10	42.58	20.88	37.38	7.57
Base Challenger	30.83	44.36	24.77	47.59	6.59
R-Zero (Iter 1)	34.27	48.06	**27.92**	51.69	9.42
R-Zero (Iter 2)	**34.92**	48.44	27.72	**53.75**	9.76
R-Zero (Iter 3)	34.64	**49.07**	27.55	51.53	**10.42**
Qwen3-8B-Base					
Base Model	34.49	49.18	28.33	51.80	8.63
Base Challenger	36.43	51.87	30.12	54.14	9.60
R-Zero (Iter 1)	37.93	53.39	31.26	57.17	9.91
R-Zero (Iter 2)	38.45	53.84	**31.58**	58.20	10.20
R-Zero (Iter 3)	**38.73**	**54.69**	31.38	**58.23**	**10.60**
OctoThinker-3B					
Base Model	12.27	26.64	10.09	10.87	1.46
Base Challenger	14.41	27.51	11.19	14.53	**4.40**
R-Zero (Iter 1)	14.93	27.76	12.21	15.72	4.05
R-Zero (Iter 2)	15.11	28.20	12.43	16.08	3.74
R-Zero (Iter 3)	**15.67**	**29.32**	**12.44**	**16.71**	4.20
OctoThinker-8B					
Base Model	16.81	32.11	13.26	20.21	1.64
Base Challenger	25.08	36.41	16.99	41.46	5.46
R-Zero (Iter 1)	26.44	37.80	19.15	**42.05**	6.77
R-Zero (Iter 2)	26.77	38.23	19.27	41.34	**8.25**
R-Zero (Iter 3)	**26.88**	**38.52**	**19.82**	40.92	**8.25**

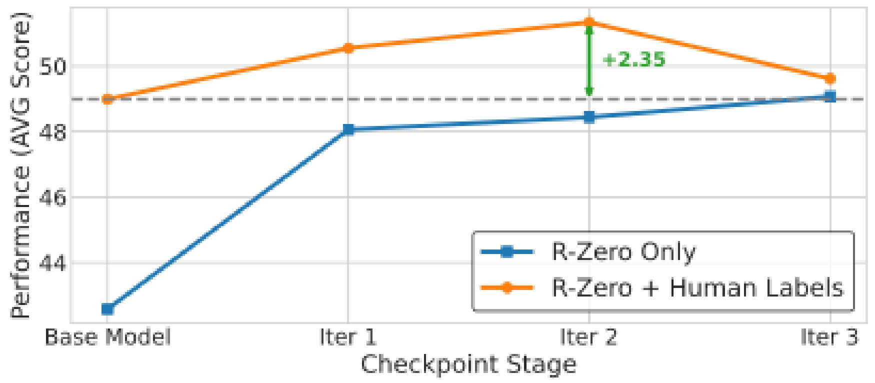
These gains are not domain-specific — they generalize beyond math and enhance core reasoning ability

- Findings:
  - R-Zero improves **all tested models** in general reasoning
  - **Qwen3-8B:** +3.81 points (34.49 → 38.73)
  - **OctoThinker-3B:** +3.65 points (12.27 → 15.67)
  - Iterative gains across 3 rounds, similar to math results
- **Takeaway:** R-Zero's math-based training **transfers to general reasoning skills**

Analysis – Ablation Study

- Removing **RL-Challenger**, **Filtering**, or **Repetition Penalty** → sharp performance drop.
- Biggest loss: without RL-Challenger (−3.7 math, −4.1 general).
- **Takeaway:** Each module is essential; Challenger RL drives curriculum quality

Method	Math AVG	General AVG
R-Zero (full)	48.06	30.41
Ablations		
⊢ w/o RL-Challenger	44.36	26.32
⊢ w/o Rep. Penalty	45.76	27.56
⊢ w/o Filtering	47.35	24.26



	Performance of Evaluated Model (vs. Ground Truth)				
	Base Model	Solver (Iter 1)	Solver (Iter 2)	Solver (Iter 3)	Pseudo-Label Acc.
$\mathcal{D}_{\text{Iter 1}}$	48.0	59.0	57.0	61.0	79.0%
$\mathcal{D}_{\text{Iter 2}}$	52.5	53.0	51.5	53.5	69.0%
$\mathcal{D}_{\text{Iter 3}}$	44.0	47.0	45.0	50.5	63.0%

Analysis – Difficulty & Synergy

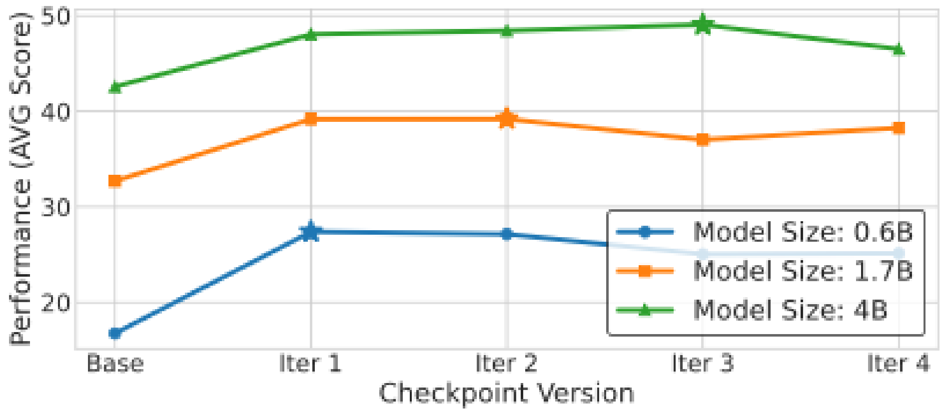
- **Difficulty Evolution:** Challenger makes tasks harder each round, but pseudo-label accuracy falls (79% → 63%)
- **Synergy with Human Labels:** Adding labeled data after R-Zero training yields **+2.35 points** over supervised baseline
- **Takeaway:** R-Zero improves difficulty handling, and works even better when combined with human labels

Analysis – Scaling & Design

- **Iteration Scaling:** Larger models delay collapse; small models degrade earlier.
- **Label Noise:** Collapse linked to declining pseudo-label accuracy (but not the sole factor).
- **Two-Model Design:** Separate Challenger & Solver sustains higher performance (49.07 vs 45.57 for Single-R-Zero).
- **Takeaway:** Bigger models and two-model design stabilize training, but collapse risk remains.

	R-Zero (ours)		Single-R-Zero	
Iteration	Performance	Pseudo-label Acc (%)	Performance	Pseudo-label Acc (%)
Iter 1	48.06	71.0	**47.31**	63.4
Iter 2	48.44	56.2	46.95	46.6
Iter 3	**49.07**	48.8	45.57	32.6
Iter 4	46.52	42.2	43.89	33.8

Iteration	Model Size		
	0.6B	1.7B	4B
Iter 1	**70.6**	69.4	71.0
Iter 2	53.4	**55.2**	56.2
Iter 3	50.8	52.2	**48.8**
Iter 4	44.0	45.2	42.2



Conclusion

- Contribution: R-Zero is the first framework to evolve reasoning LLMs with no external data
- Impact: Moves toward more autonomous & scalable AI training
- Limitations
  - Works best in domains with objectively verifiable answers (math)
  - Remains challenge in open-ended domains
- Future Directions
  - Improve label quality
  - Extend to broader reasoning
  - Prevent long-term collapse

Thank you!