

Demystifying LLM Reasoning through U-statistics Theory

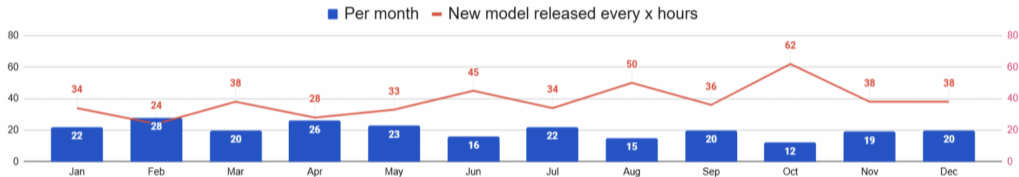
Chengchun Shi

Associate Professor of Data Science
London School of Economics and Political Science

Number of LLMs Per Month (2024)

New major models released per month/x hours

LifeArchitect.ai/models (data from LifeArchitect.ai/models-table)

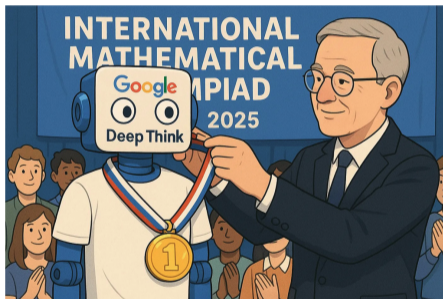


New derivative models released per month/x minutes

LifeArchitect.ai/models (data from Hugging Face)



The Year of 2025



nature

[Explore content](#) ▾ [About the journal](#) ▾ [Publish with us](#) ▾ [Subscribe](#)

[nature](#) > [editorials](#) > [article](#)

EDITORIAL | 17 September 2025

Bring us your LLMs: why peer review is good for AI models

Deepseek's R1 model has been peer reviewed. Others should follow the firm's example.

DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models

Zhihong Shao^{1,2*†}, Peiyi Wang^{1,3*†}, Qihao Zhu^{1,3*†}, Runxin Xu¹, Junxiao Song¹
Xiao Bi¹, Haowei Zhang¹, Mingchuan Zhang¹, Y.K. Li¹, Y. Wu¹, Daya Guo^{1*}

¹DeepSeek-AI, ²Tsinghua University, ³Peking University

[nature](#) > [articles](#) > [article](#)

Article | [Open access](#) | Published: 17 September 2025

DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning

[Daya Guo](#), [Dejian Yang](#), [Haowei Zhang](#), [Junxiao Song](#), [Peiyi Wang](#), [Qihao Zhu](#), [Runxin Xu](#), [Ruoyu Zhang](#), [Shirong Ma](#), [Xiao Bi](#), [Xiaokang Zhang](#), [Xingkai Yu](#), [Yu Wu](#), [Z. F. Wu](#), [Zhibin Gou](#), [Zhihong Shao](#), [Zhuoshu Li](#), [Ziyi Gao](#), [Aixin Liu](#), [Bing Xue](#), [Bingxuan Wang](#), [Bochao Wu](#), [Bei Feng](#), [Chengda Lu](#), ... [Zhen Zhang](#) [+ Show authors](#)

[Nature](#) **645**, 633–638 (2025) | [Cite this article](#)



Since then, 100+ xPO variants have been proposed

Rollout Selection & Bias Correction:

- **SRPO** (Zhang et al., 2025c): history resampling
- **DAPO** (Yu et al., 2025): dynamic sampling
- **Dr.GRPO** (Liu et al., 2025): mitigates length bias
- **OPO** (Hao et al., 2025): optimal baseline to reduce gradient variance

Reward Shaping & Advantage Estimation:

- **EMPO** (Zhang et al., 2025b): fully unsupervised / minimizes predictive entropy
- **AAPO** (Xiong et al., 2025a): advantage momentum
- **BNPO** (Xiao et al., 2025): adaptively normalizes rewards via Beta distribution
- **SEED-GRPO** (Chen et al., 2025): : uses semantic entropy to scale updates by uncertainty
- **GRPO-lead** (Zhang & Zuo, 2025): length-dependent accuracy, explicit penalties, difficulty-aware reweighting

Efficiency-Driven Methods:

- **CPPO** (Lin et al., 2025): pruning low-advantage completions
- **S-GRPO** (Dai et al., 2025b): early exit to cut redundancy
- **Ada-GRPO** (Wu et al., 2025): adaptive reasoning formats
- **GVPO** (Zhang et al., 2025a): analytical KL-constrained weighting
- **GRPO- λ** (Dai et al., 2025a): ddynamic switch between length-penalized vs length-agnostic rewards to avoid collapse

Gap between theory and practice

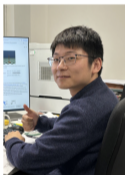
Q1 *Why is GRPO so effective?*

Q2 *What is the rationale for using the group mean to approximate the value function?*

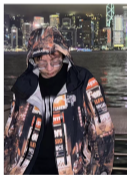
Q3 *Can we provide finite-sample or asymptotic analyses regarding its convergence?*

Q4 *What is the optimal group size?*

Demystifying GRPO: Its Policy Gradient is a U-Statistic



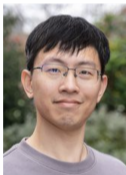
Hongyi Zhou
Tsinghua



Kai Ye
LSE



Erhan Xu
LSE



Jin Zhu
Birmingham



Ying Yang
Tsinghua



Shijin Gong
USTC

Background

1. Reinforcement Learning
2. LLM Reasoning
3. U-statistics

Background

1. Reinforcement Learning

2. LLM Reasoning

3. U-statistics

Reinforcement Learning (RL)

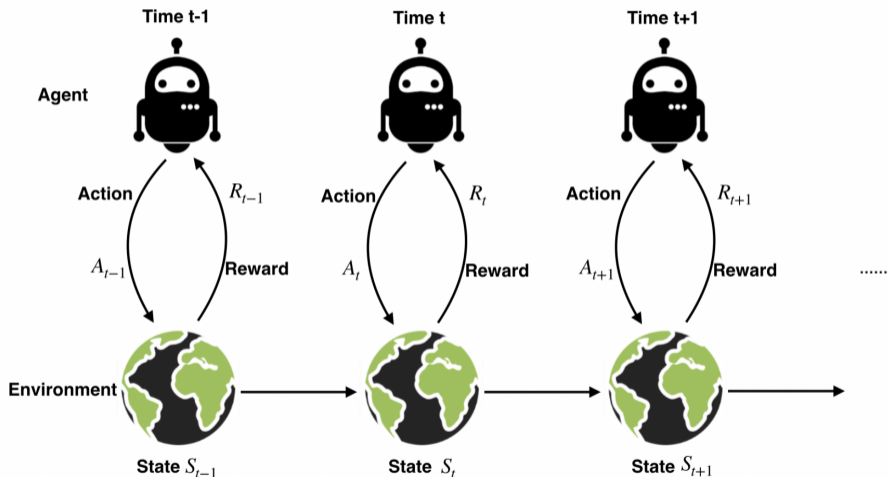
Andrew Barto and Richard Sutton Receive A.M. Turing Award



The scientists received computing's highest honor for developing the theoretical foundations of reinforcement learning, a key method for many types of AI.

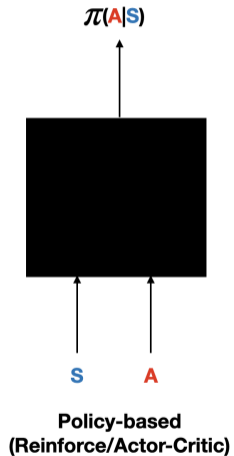
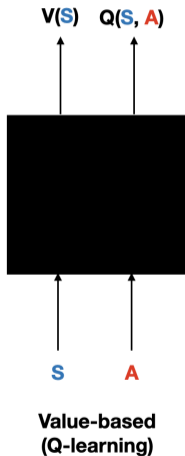
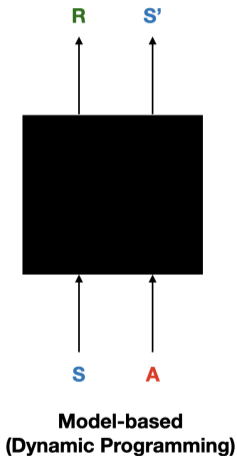


Reinforcement Learning (Cont'd)

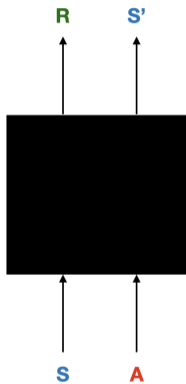


Objective: find an optimal policy that maximizes the cumulative reward

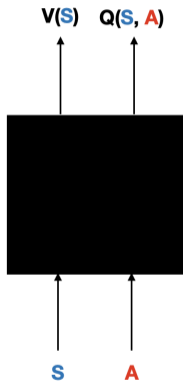
Three Types of RL Algorithms



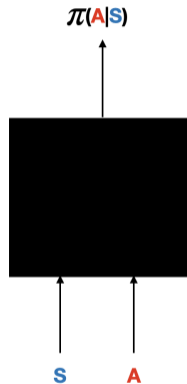
Three Types of RL Algorithms



Model-based
(Dynamic Programming)



Value-based
(Q-learning)



Policy-based
(Reinforce/Actor-Critic)

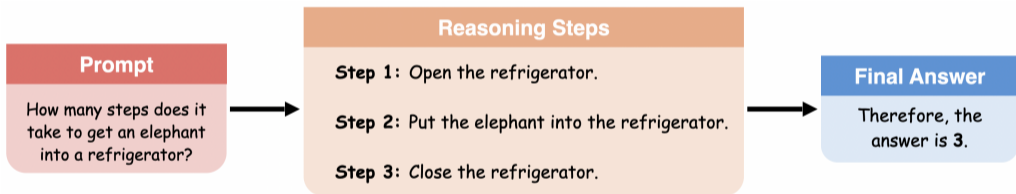
Background

1. Reinforcement Learning

2. LLM Reasoning

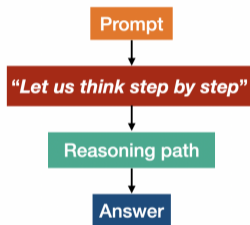
3. U-statistics

LLM Reasoning

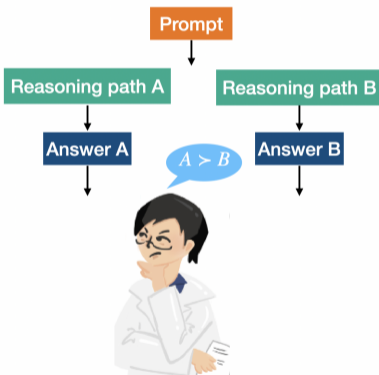


Approach I: Chain-of-Thought Prompting

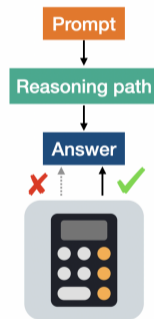
Chain-of-Thought Prompting (Wei et al. 2022)



Reinforcement Learning from Human Feedback

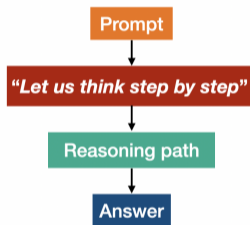


Reinforcement Learning from Verifiable Rewards

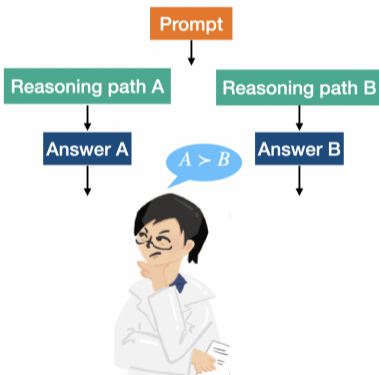


Approach II: RLHF

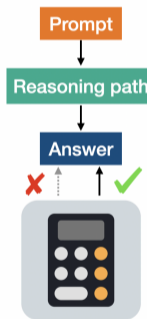
Chain-of-Thought Prompting (Wei et al. 2022)



Reinforcement Learning from Human Feedback



Reinforcement Learning from Verifiable Rewards



Proximal policy optimization (Ouyang et al., 2022)

1) Policy Model π_θ

The LLM we update (the “actor”)

2) Reward Model $r(x, y)$

Learn from human feedback

3) Value Function $V^{\pi_\theta}(x)$

Baseline we use to calculate the advantage function for variance reduction

4) Reference Policy π_{ref}

To prevent reward hacking

$$r(x, y) - \beta \text{KL}(\pi_\theta(\cdot | x) \| \pi_{ref}(\cdot | x))$$

```
def rlhf_training(policy_model, reward_model, value_model, prompts):
    ref_policy = freeze(copy(policy_model))
    for step in range(num_steps):
        experiences = []
        for prompt in prompts:
            # Generate response
            response = policy_model.generate(prompt)
            # Get reward from reward model
            reward = reward_model(prompt, response)
            logp_old = policy_model.logprob(prompt, response)
            # Add KL penalty (stay close to reference policy)
            kl_penalty = kl_divergence(policy_model, ref_policy)
            final_reward = reward - beta * kl_penalty
            experiences.append((prompt, response,
                               final_reward, logp_old))
            # Advantage (usually estimated by GAE)
            adv = compute_advantages(experiences, value_model)
            experience.append(adv)
        # PPO update
        for _ in range(ppo_epochs):
            ppo_step(policy_model, experiences)
    return policy_model
```

PPO (Cont'd)

1) Policy Model π_θ

The LLM we update (the “actor”)

2) Reward Model $r(x, y)$

Challenge I: Human labelling is costly in reasoning

3) Value Function $V^{\pi_\theta}(x)$

Challenge II: Estimating and storing a value function is computationally inefficient

4) Reference Policy π_{ref}

To prevent reward hacking

$$r(x, y) - \beta \text{KL}(\pi_\theta(\cdot | x) \| \pi_{ref}(\cdot | x))$$

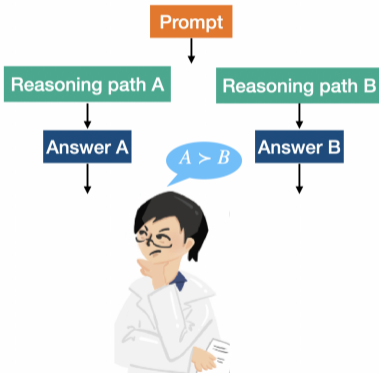
```
def rlhf_training(policy_model, reward_model, value_model, prompts):
    ref_policy = freeze(copy(policy_model))
    for step in range(num_steps):
        experiences = []
        for prompt in prompts:
            # Generate response
            response = policy_model.generate(prompt)
            # Get reward from reward model
            reward = reward_model(prompt, response)
            logp_old = policy_model.logprob(prompt, response)
            # Add KL penalty (stay close to reference policy)
            kl_penalty = kl_divergence(policy_model, ref_policy)
            final_reward = reward - beta * kl_penalty
            experiences.append((prompt, response,
                               final_reward, logp_old))
            # Advantage (usually estimated by GAE)
            adv = compute_advantages(experiences, value_model)
            experience.append(adv)
        # PPO update
        for _ in range(ppo_epochs):
            ppo_step(policy_model, experiences)
    return policy_model
```

Approach III: RLVR

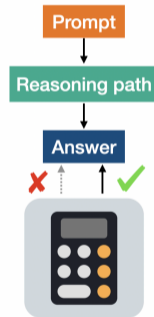
Chain-of-Thought Prompting (Wei et al. 2022)



Reinforcement Learning from Human Feedback



Reinforcement Learning from Verifiable Rewards



RLVR (Lambert et al., 2024)

Motivation

Problem with RLHF:

- Human preferences are subjective
- Reward model can be inaccurate
- Hard to verify correctness

RLVR Solution:

- Use objective, verifiable rewards
- Automatic verification
- Ground truth available
- Scalable and reliable

Key Principle

Instead of "Is this good?" ask "Is this correct?"

Suitable for:

- **Code:** Run unit tests ✓
- **Math:** Check answer ✓
- **Logic:** Verify proof ✓

$$R_{\text{RLVR}}(x, y) = \begin{cases} +1 & \text{if correct} \\ 0 & \text{if wrong} \end{cases}$$

Pseudocode

- **Initialization:** θ arbitrary
- **For** $i = 1, 2, \dots, n$ **do:**

Sample an prompt X

Generate a completion Y (reasoning trace + response) using policy π_θ

Obtain the verifiable reward $Z = r(X, Y)$:

$$\theta \leftarrow \theta + \eta \nabla_\theta \log(\pi_\theta(Y|X))Z$$

end for

return θ

Variance reduction using a baseline

- **Initialization:** θ arbitrary
- **For** $i = 1, 2, \dots, n$ **do:**

Sample an prompt X

Generate a completion Y using policy π_θ

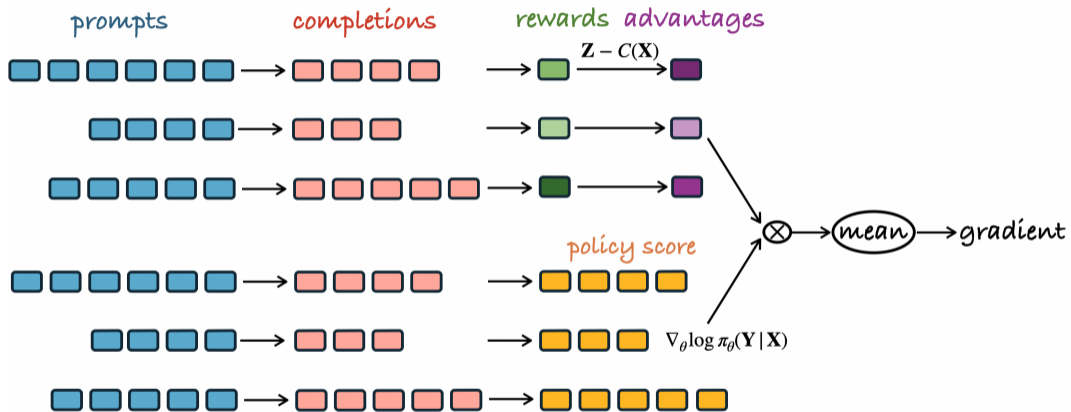
Obtain the verifiable reward $Z = r(X, Y)$:

$$\theta \leftarrow \theta + \eta \nabla_\theta \log(\pi_\theta(Y|X))[Z - C(X)]$$

end for

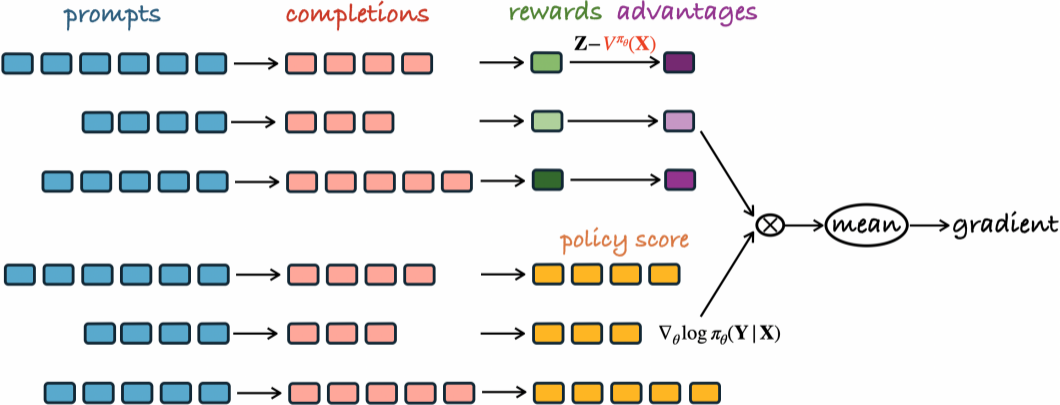
return θ

A minibatch version



$$\theta \leftarrow \theta + \eta \text{ gradient}$$

An oracle version



$$\theta \leftarrow \theta + \eta \text{ gradient}$$

Background

1. Reinforcement Learning

2. LLM Reasoning

3. U-statistics

Second-order U-statistics

- $\{\mathbf{X}_i\}_{i=1}^n$ denotes a sequence of i.i.d. random vectors
- Given a symmetric kernel function \mathbf{h}
- A second-order U-statistic is defined as

$$\mathbf{U} = \frac{1}{n(n-1)} \sum_{i \neq j} \mathbf{h}(\mathbf{X}_i, \mathbf{X}_j).$$

- Hoeffding decomposition:

$$\mathbf{U} = \mathbf{h}_0 + \underbrace{\frac{2}{n} \sum_{i=1}^n [\mathbf{h}_1(\mathbf{X}_i) - \mathbf{h}_0]}_{\text{first-order term } O_p(n^{-1/2})} + \underbrace{\frac{1}{n(n-1)} \sum_{i \neq j} [\mathbf{h}(\mathbf{X}_i, \mathbf{X}_j) - \mathbf{h}_1(\mathbf{X}_i) - \mathbf{h}_1(\mathbf{X}_j) + \mathbf{h}_0]}_{\text{second-order term } O_p(n^{-1})},$$

where $\mathbf{h}_0 = \mathbb{E}[\mathbf{h}(\mathbf{X}_1, \mathbf{X}_2)]$ is the expectation of the kernel and $\mathbf{h}_1(\mathbf{x}) = \mathbb{E}[\mathbf{h}(\mathbf{x}, \mathbf{X}_2)]$ denotes the first-order projection.

Bring it all together: GRPO

Motivation

Problem with PPO:

- a separate value model/critic increases GPU memory.
- a poorly learned value model makes advantages noisy, destabilizing updates.

GRPO Solution:

- No value network needed
- Uses group-based advantages

Key Idea

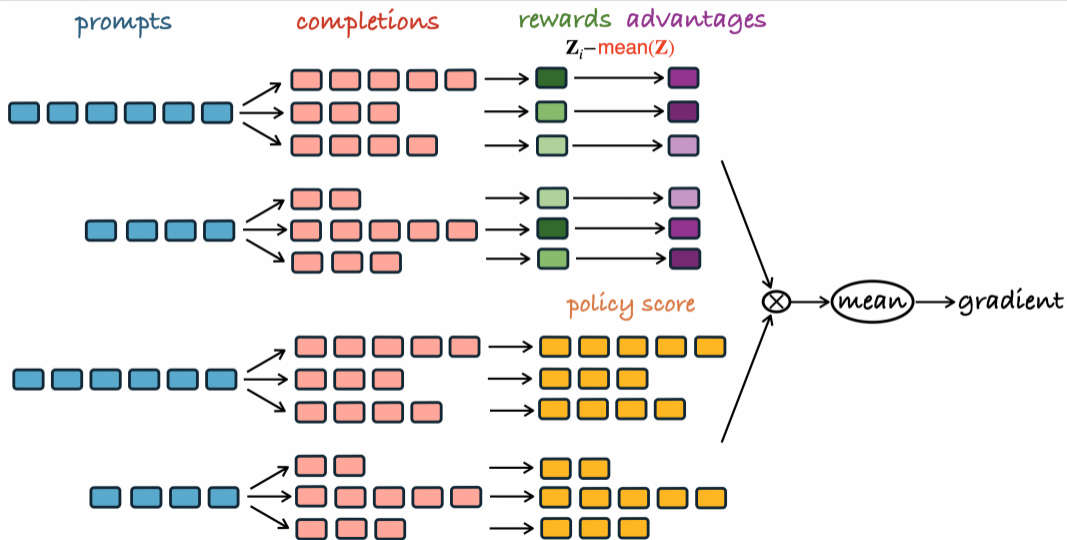
Instead of comparing to a baseline **by model**, compare responses **within a group**:

1. Generate **multiple responses** for same prompt
2. Score all responses with reward model
3. Use **group statistics** for advantage
4. Update policy based on relative quality

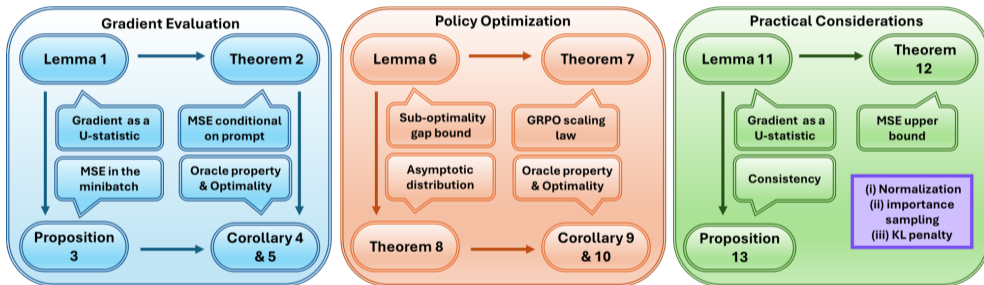
$$\text{Advantage}_i = Z_i - \text{mean}(Z)$$

No separate value network needed!

GRPO-type algorithm



Demystify GRPO: Our results



Demystify GRPO (Cont'd)

Q1 *Why is GRPO so effective?*

Q2 *What is the rationale for using the group mean to approximate the value function?*

Q3 *Can we provide finite-sample or asymptotic analyses regarding its convergence?*

Q4 *What is the optimal group size?*

GRPO's gradient is a U-statistic

- G = no. of sampled completions per prompt.
- GRPO-type gradient:

$$\hat{g}_{\text{GRPO}}(\mathbf{x}; \theta) = \frac{1}{G-1} \sum_{g=1}^G \underbrace{\nabla_{\theta} \log \pi_{\theta}(\mathbf{Y}^{(g)} | \mathbf{x})}_{\text{policy score}} \underbrace{[\mathbf{Z}^{(g)} - \text{mean}(\mathbf{Z})]}_{\text{advantage}}.$$

Lemma

$\hat{g}_{\text{GRPO}}(\mathbf{x}; \theta)$ can be written as a second-order U-statistic.

Proof

- Each individual term in $\hat{g}_{\text{GRPO}}(\mathbf{x}; \boldsymbol{\theta})$ equals

$$\frac{G}{G-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{Y}^{(g)} | \mathbf{x}) [\mathbf{Z}^{(g)} - \text{mean}(\mathbf{Z})] = \frac{1}{G-1} \sum_{k \neq g} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{Y}^{(g)} | \mathbf{x}) [\mathbf{Z}^{(g)} - \mathbf{Z}^{(k)}]$$

- **Averaging** over all $g \in \{1, \dots, G\}$ & applying **symmetrization**

$$\hat{g}_{\text{GRPO}}(\mathbf{x}; \boldsymbol{\theta}) = \binom{G}{2}^{-1} \sum_{1 \leq i < j \leq G} \frac{1}{2} [\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{Y}^{(i)} | \mathbf{x}) - \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{Y}^{(j)} | \mathbf{x})] (\mathbf{Z}^{(i)} - \mathbf{Z}^{(j)})$$

According to Hoeffding decomposition

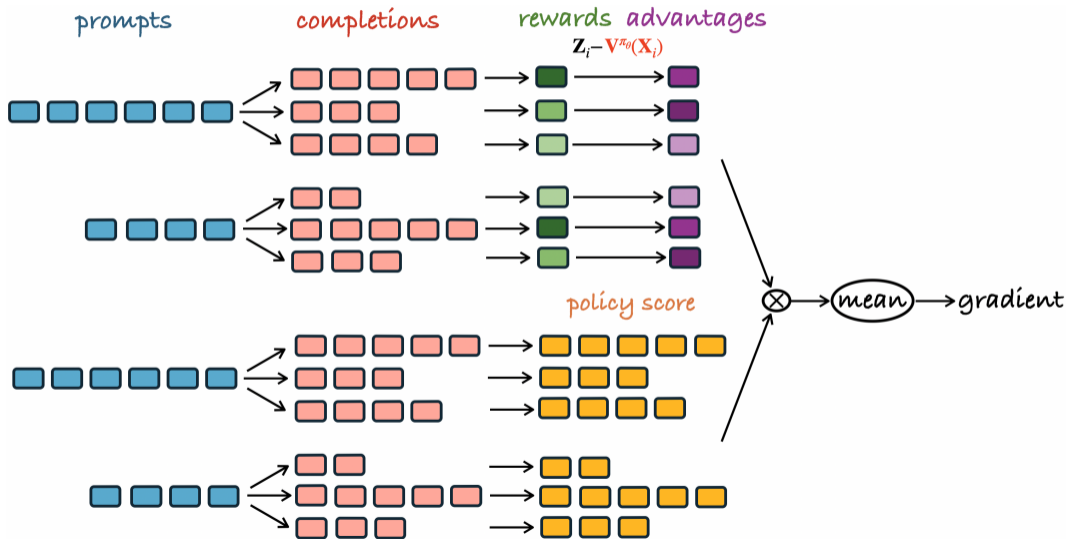
$$\widehat{g}_{\text{GRPO}}(\mathbf{x}; \boldsymbol{\theta}) = \text{true gradient at } \mathbf{x} + \underbrace{\widehat{g}_{\text{oracle}}(\mathbf{x}; \boldsymbol{\theta}) - \mathbb{E}[\widehat{g}_{\text{oracle}}(\mathbf{x}; \boldsymbol{\theta})]}_{\text{first-order term } O_p(\mathbf{G}^{-1/2})} + \underbrace{\text{second-order term}}_{O_p(\mathbf{G}^{-1})}$$

- B denotes the number of prompts
- Consider the minibatch version $\widehat{g}_{\text{GRPO}}(\boldsymbol{\theta}) = \frac{1}{B} \sum_{b=1}^B \widehat{g}_{\text{GRPO}}(\mathbf{x}^{(b)}; \boldsymbol{\theta})$

Theorem

$$\text{MSE}(\widehat{g}_{\text{GRPO}}(\boldsymbol{\theta})) = \text{MSE}(\widehat{g}_{\text{oracle}}(\boldsymbol{\theta})) + O\left(\frac{\mathbb{E}\|\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{Y}|\mathbf{X})\|^2}{BG^2}\right)$$

Oracle algorithm



Demystify GRPO (Cont'd)

Q1 *Why is GRPO so effective?*

Q2 *What is the rationale for using the group mean to approximate the value function?*

A2 Use of group mean provides a **first-order approximation** to the gradient of the **oracle algorithm** with access to the value function

Q3 *Can we provide finite-sample or asymptotic analyses regarding its convergence?*

Q4 *What is the optimal group size?*

Demystify GRPO (Cont'd)

Q1 *Why is GRPO so effective?*

Q2 *What is the rationale for using the group mean to approximate the value function?*

Q3 *Can we provide finite-sample or asymptotic analyses regarding its convergence?*

Q4 *What is the optimal group size?*

Demystify GRPO (Cont'd)

Corollary (Oracle property)

*As $G \rightarrow \infty$, both (i) **MSE** of the GRPO-type gradient and (ii) **suboptimality gap** of its induced policy are asymptotically equivalent to those of the **oracle** algorithm.*

Corollary (Optimality)

*As $G \rightarrow \infty$, both (i) the MSE of the GRPO-type gradient and (ii) the suboptimality gap of its induced policy are asymptotically no larger than those of **any** algorithm using a generic baseline $\mathbf{C}(\mathbf{X})$. In particular, the gradient MSE is strictly smaller than that of the **vanilla** algorithm with zero baseline, under mild conditions.*

In DeepSeekMath, $G = \mathbf{64}$, which is sufficiently large to approximate the asymptotics.

Experiment I: Gradient evaluation

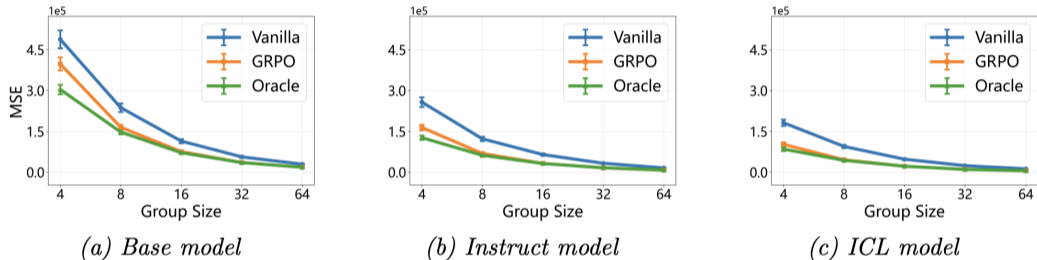


Figure 4: MSEs of three policy gradient estimators (vanilla, GRPO-type, and oracle) under three model configurations (base, instruct and in-context learning (ICL)) for different group sizes. Error bars represent 95% confidence intervals of the empirically estimated MSEs.

Demystify GRPO (Cont'd)

Q1 *Why is GRPO so effective?*

A1 Because it achieves **oracle** performance without estimating the value function and is asymptotically **optimal** in both **MSE** and **suboptimality gap**.

Q2 *What is the rationale for using the group mean to approximate the value function?*

Q3 *Can we provide finite-sample or asymptotic analyses regarding its convergence?*

Q4 *What is the optimal group size?*

Demystify GRPO (Cont'd)

Q1 *Why is GRPO so effective?*

Q2 *What is the rationale for using the group mean to approximate the value function?*

Q3 *Can we provide finite-sample or asymptotic analyses regarding its convergence?*

A3 **Yes** – this is exactly what we establish in the paper. In particular, we establish parameter consistency and derive the asymptotic distribution of the suboptimality gap (a mixture of χ^2 random variables) without requiring parameter identifiability.

Q4 *What is the optimal group size?*

Demystify GRPO (Cont'd)

Q1 *Why is GRPO so effective?*

Q2 *What is the rationale for using the group mean to approximate the value function?*

Q3 *Can we provide finite-sample or asymptotic analyses regarding its convergence?*

Q4 *What is the optimal group size?*

A4 **We derive a scaling law**

- (i) delineates how GRPO's performance depends on the group size
- (ii) identifies the optimal group size that maximizes its performance

GRPO scaling law

Theorem

GRPO's sub-optimality upper bound depend on (i) the number of sampled prompts B and (ii) the number of sampled completions G per prompt through the following:

$$\frac{c_1}{B} + \frac{c_2}{BG} + \frac{c_3}{BG^2},$$

*where c_1, c_2, c_3 depend only on the **data generating process and geometry of the policy space**.*

Under a fixed sampling budget $N = BG$ per iteration or a total sampling budget $\mathbf{N} = nBG$, the optimal group size G^ that minimizes the upper bound is:*

$$G^* = \sqrt{\frac{c_3}{c_1}}.$$

Universality of G^*

The optimal group size G^* is **universal**:

- independent of the **budget** N
- independent of the **number of iterations** n
- independent of the **learning rate schedule**

depending solely on the **data generating process** and **geometry of the policy space**.

Experiment II: Universality across training iterations

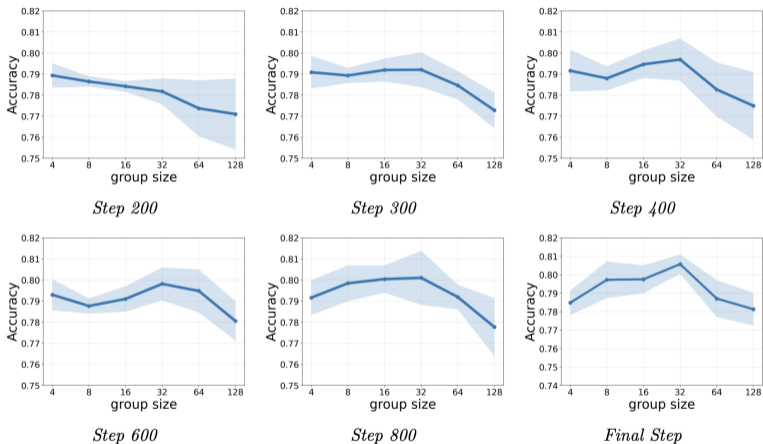


Figure 5: Test accuracy of GRPO-fine-tuned models at different training steps with a fixed sampling budget of $N = B \times G = 1024$ per prompt. Both training and evaluation are conducted on GSM8K. Each curve shows accuracy as a function of the group size $G \in \{4, 8, 16, 32, 64, 128\}$. Results are averaged over five independent runs, with shaded regions visualizing 95% confidence bands.

Experiment III: Universality across sampling budgets

Table 2: Test accuracy of GRPO-fine-tuned models at the final training step. Each row reports accuracy as a function of the group size $G \in 4, 8, 16, 32, 64, 128$ with a fixed sampling budget per prompt; the sampling budget varies across rows. Both training and evaluation are conducted on MATH. Due to the high computational cost of training a 7B model, results are reported from a single run, with the highest accuracy highlighted in bold.

Sampling budget	Group size G					
	4	8	16	32	64	128
1024	0.7677	0.7491	0.7753	0.7627	0.7817	0.7743
2048	0.7703	0.7679	0.7697	0.7793	0.7819	0.7665
4096	0.7691	0.7713	0.7701	0.7673	0.7703	0.7757

Thank You!

😊 Papers can be found on my personal website

`callmespring.github.io`