IMPERIAL

NLP Reading Group

June 3rd 2025

“GRPO & Deepseek-R1”

Group Relative Policy Optimisation (GRPO)

Group Relative Policy Optimisation (GRPO)

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

Zhihong Shao[1,2*†], Peiyi Wang[1,3*†], Qihao Zhu[1,3*†], Runxin Xu[1], Junxiao Song[1]
Xiao Bi[1], Haowei Zhang[1], Mingchuan Zhang[1], Y.K. Li[1], Y. Wu[1], Daya Guo[1*]

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

{zhihongshao,wangpeiyi,zhuqh,guoday}@deepseek.com
https://github.com/deepseek-ai/DeepSeek-Math

27th April **2024**

IMPERIAL

GRPO or How to get a really good reasoning model?

Main Contributions of GRPO paper:

1. DeepSeekMath-Corpus (120B tokens) for continued pre-training
2. GRPO algorithm & analysis (144K training samples)
3. SOTA 7B model on math reasoning (given its size).
4. (Special mention) SFT Instruction tuning dataset collection; 776K examples.

GRPO: Some motivation

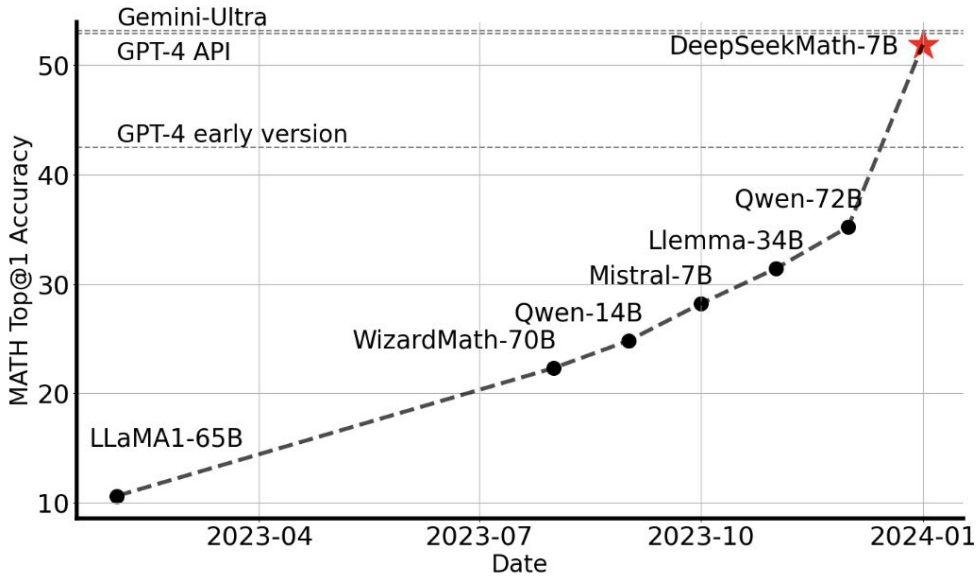


Figure 1 | Top1 accuracy of open-source models on the competition-level MATH benchmark (Hendrycks et al., 2021) without the use of external toolkits and voting techniques.

Part 1: Dataset Curation

GRPO: Part 1: Pre-training Corpus

Pre-training Corpus:

1. Filtered from Common Crawl 4.0 (a large web crawl dataset)
2. Classifier is training on **fastText** !!! (This is from 2016, based on static word-vectors)
3. Iteratively filtering dataset and extending the “seed” dataset to allow for larger math seed corpus.
4. In total 120B tokens (e.g. Modern LLMs train on 3T - 15T, or more) so this is 5% of the total pre-training corpus (i.e. quite significant).
5. A total of 3 rounds of filtering (in last round they had captured already 98% compared to next round).

GRPO: Part 1: Pretraining Corpus

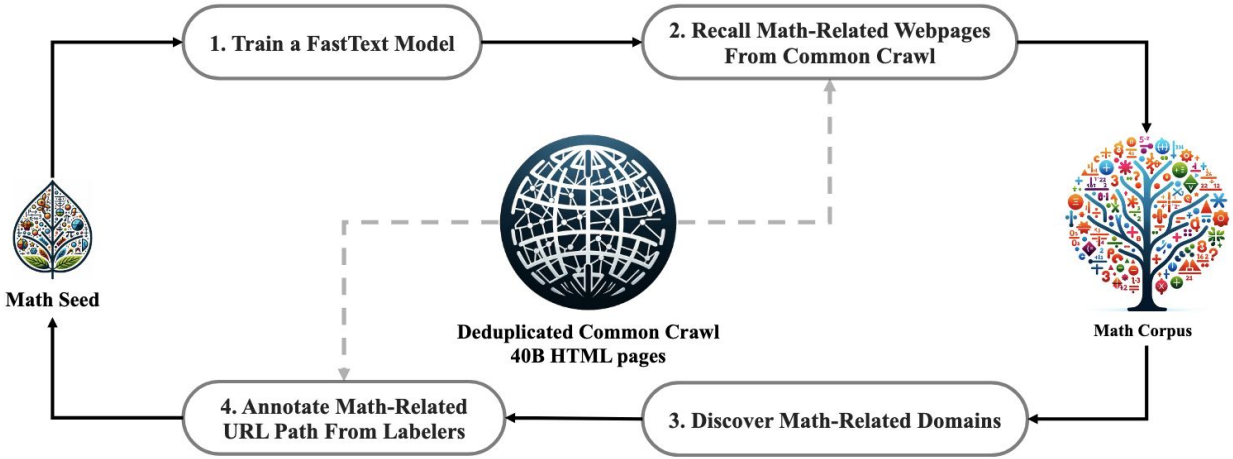


Figure 2 | An iterative pipeline that collects mathematical web pages from Common Crawl.

Part 2: GRPO Algorithm

GRPO: Part 2: GRPO Algorithm

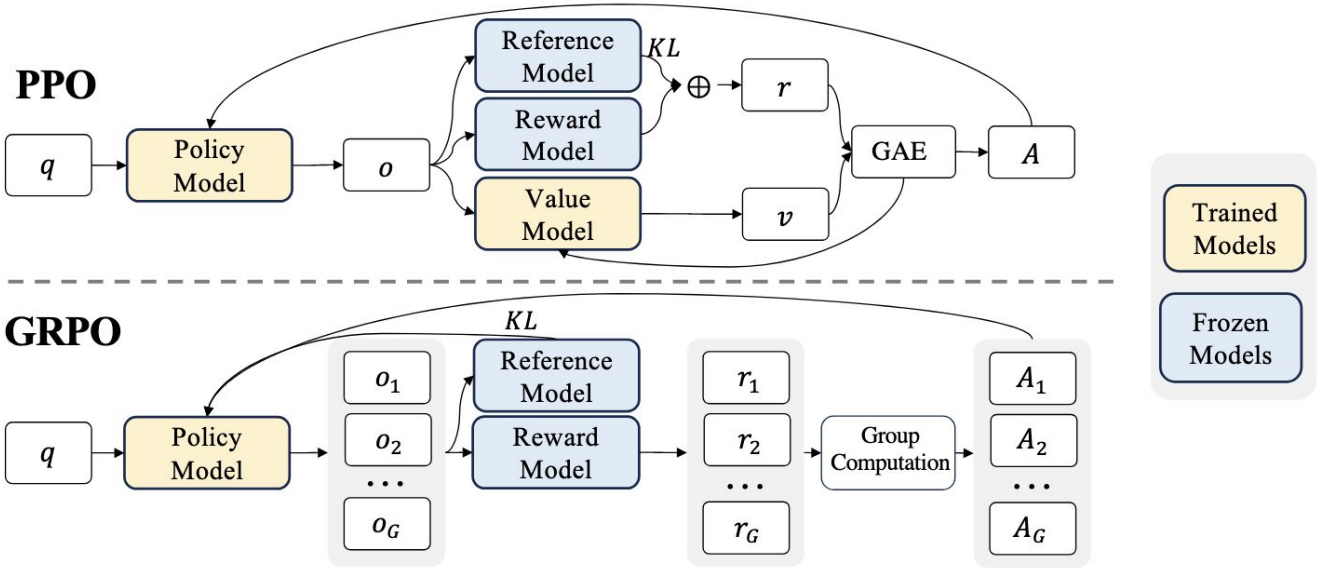


Figure 4 | Demonstration of PPO and our GRPO. GRPO foregoes the value model, instead estimating the baseline from group scores, significantly reducing training resources.

GRPO: Part 2: GRPO Algorithm

Starting point Policy Optimisation Methods & Proximal Policy Optimisation Algorithm (**2017**)

1. In RL one wants to optimise E[R]; one either does this via Q-learning (or value-based RL); or policy optimisation (policy-based RL).
2. Classical unbiased estimate for the gradient of $\nabla E[R] \sim \nabla \theta \log \pi(a_t|s_t; \theta) R_t$ (from REINFORCE)
3. A better unbiased estimate is: $\nabla E[R] \sim \nabla \theta \log \pi(a_t|s_t; \theta) (R_t - b_t(s_t))$; where b_t is a baseline, often the Value function. $(R_t - b_t(s_t)) = A_t$ is called the **Advantage**.

GRPO: Part 2: GRPO Algorithm

Proximal Policy Optimisation (PPO) is crucial in modern LLMs (via RLHF)

1. The key innovation of GRPO is to replace the Value function from PPO with an approximation from averages (this massively reduces memory constraints; as Value function is often as big as the Policy; i.e 2x LLMs in memory).
2. However, knowing that this comes from the classical Policy Gradient Estimation theory, suddenly GRPO makes perfect sense!

GRPO: Part 2: GRPO Algorithm

The GRPO algorithm:

1. **$\nabla E[R] \sim \nabla \theta \log \pi(a_t|s_t; \theta) (A_t)$ ***
2. Where, $A_t = R_t - b_t(s_t)$
3. Where b_t = Average of the current sample from the policy.

*=technically GRPO also has a KL divergence term to not allow the policy to deviate too much from the reference policy (which is the original LLM usually).

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

IMPERIAL

GRPO: Part 2: GRPO Algorithm

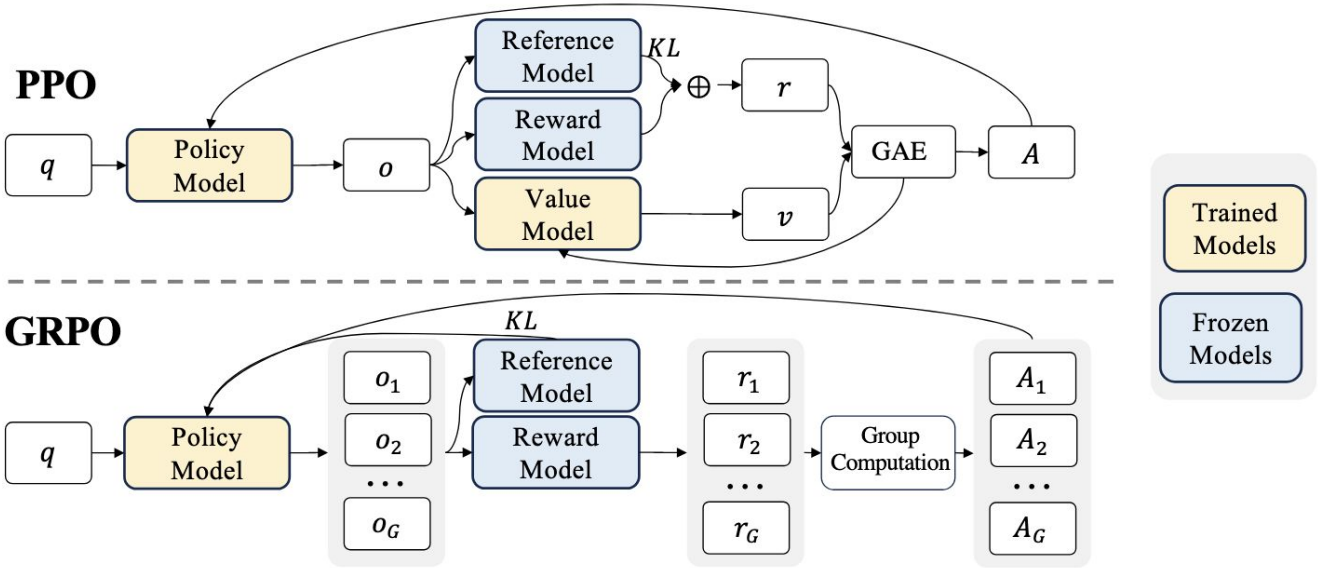


Figure 4 | Demonstration of PPO and our GRPO. GRPO foregoes the value model, instead estimating the baseline from group scores, significantly reducing training resources.

Part 3: Experiments & Results

GRPO: Part 3: Experiments & Results

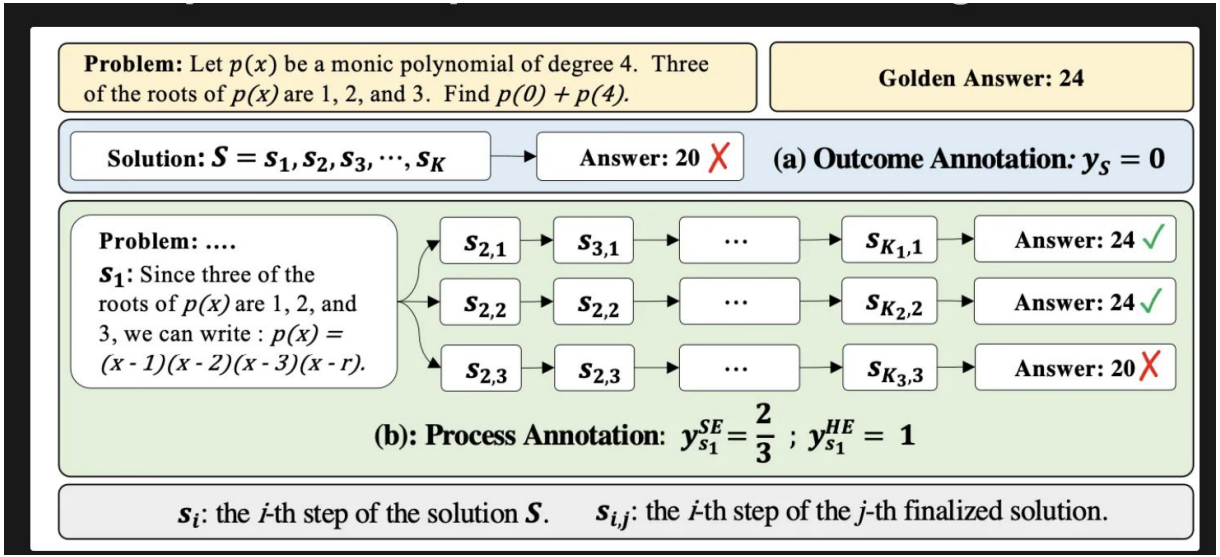
RL Setup

1. Outcome supervision
2. Process supervision (how can they evaluate intermediate steps?)
3. Iterative RL (re-training reward model)

GRPO: Part 3: Experiments & Results

Process Supervision

1. Math-Shepard Paper



GRPO: Part 3: Experiments & Results

Hyper-params

1. 64 outputs per question, 1024 Max-tokens, 1024 batch-size (confused what batch-size means. Maybe 16 questions per update?)
2. 144K Questions from SFT dataset.

GRPO: Part 3: Experiments & Results

Evaluation datasets:

1. Loads of datasets:
    a. Math, Math with Tools, Formal Maths, General reasoning benchmarking (e.g. MMLU)

2. GSM8K & MATH500

GRPO: Part 3: Experiments & Results

Model	Size	English Benchmarks		Chinese Benchmarks	
		GSM8K	MATH	MGSM-zh	CMATH
Chain-of-Thought Reasoning					
Closed-Source Model					
Gemini Ultra	-	94.4%	53.2%	-	-
GPT-4	-	92.0%	52.9%	-	86.0%
Inflection-2	-	81.4%	34.8%	-	-
GPT-3.5	-	80.8%	34.1%	-	73.8%
Gemini Pro	-	86.5%	32.6%	-	-
Grok-1	-	62.9%	23.9%	-	-
Baichuan-3	-	88.2%	49.2%	-	-
GLM-4	-	87.6%	47.9%	-	-
Open-Source Model					
InternLM2-Math	20B	82.6%	37.7%	-	-
Qwen	72B	78.9%	35.2%	-	-
Math-Shepherd-Mistral	7B	84.1%	33.0%	-	-
WizardMath-v1.1	7B	83.2%	33.0%	-	-
DeepSeek-LLM-Chat	67B	84.1%	32.6%	74.0%	80.3%
MetaMath	70B	82.3%	26.6%	66.4%	70.9%
SeaLLM-v2	7B	78.2%	27.5%	64.8%	-
ChatGLM3	6B	72.3%	25.7%	-	-
WizardMath-v1.0	70B	81.6%	22.7%	64.8%	65.4%
DeepSeekMath-Instruct	7B	82.9%	46.8%	73.2%	84.6%
DeepSeekMath-RL	7B	**88.2%**	**51.7%**	**79.6%**	**88.8%**

IMPERIAL

GRPO: Part 3: Experiments & Results

Training Setting	Training Tokens			w/o Tool Use			w/ Tool Use	
	General	Code	Math	GSM8K	MATH	CMATH	GSM8K+Python	MATH+Python
No Continual Training	–	–	–	2.9%	3.0%	12.3%	2.7%	2.3%
Two-Stage Training								
Stage 1: General Training	400B	–	–	2.9%	3.2%	14.8%	3.3%	2.3%
Stage 2: Math Training	–	–	150B	19.1%	14.4%	37.2%	14.3%	6.7%
Stage 1: Code Training	–	400B	–	5.9%	3.6%	19.9%	12.4%	10.0%
Stage 2: Math Training	–	–	150B	**21.9%**	**15.3%**	**39.7%**	17.4%	9.4%
One-Stage Training								
Math Training	–	–	150B	20.5%	13.1%	37.6%	11.4%	6.5%
Code & Math Mixed Training	–	400B	150B	17.6%	12.1%	36.3%	**19.7%**	**13.5%**

Table 6 | Investigation of how code affects mathematical reasoning under different training settings. We experiment with DeepSeek-LLM 1.3B, and evaluate its mathematical reasoning performance without and with tool use via few-shot chain-of-thought prompting and few-shot program-of-thought prompting, respectively.

Part 4: Insights

GRPO: Part 4: Insights

Insights

1. RL enhances Maj@K performance but not Pass@K
2. => **it seems that the improvement is attributed to boosting the correct response from TopK rather than the enhancement of fundamental capabilities**

=> The authors present that this is the biggest area of interest for future work. (I.e. How can one create more generalisation using RL).

GRPO: Part 4: Insights

Reward Function Reward function is the source of the training signal. In RL, the reward function is usually the neural reward model. We think there exist three important directions for reward models: 1) **How to enhance the generalization ability of the reward model.** The reward model must be effectively generalized to handle out-of-distribution questions and advanced decoding outputs; otherwise, reinforcement learning may merely stabilize the distribution of LLMs rather than improve their fundamental capabilities; 2) **How to reflect the uncertainty of reward model.** The uncertainty could potentially act as a linking bridge between the weak reward model and the weak-to-strong learning algorithms; 3) **How to efficiently build high-quality process reward models** that can provide fine-grained training signals for the reasoning process (Lightman et al., 2023; Wang et al., 2023b).

DeepSeek-R1

DeepSeek-R1

DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning

DeepSeek-AI

research@deepseek.com

22nd January **2025**

IMPERIAL

DeepSeek-R1

Main Contributions of DeepSeek-R1 paper:

1. Large-scale RL improves models massively.
2. RL alone can get you amazing results.
3. GRPO can be used without training a reward-model!
4. Distillation into smaller models yields powerful smaller models.

DeepSeek-R1

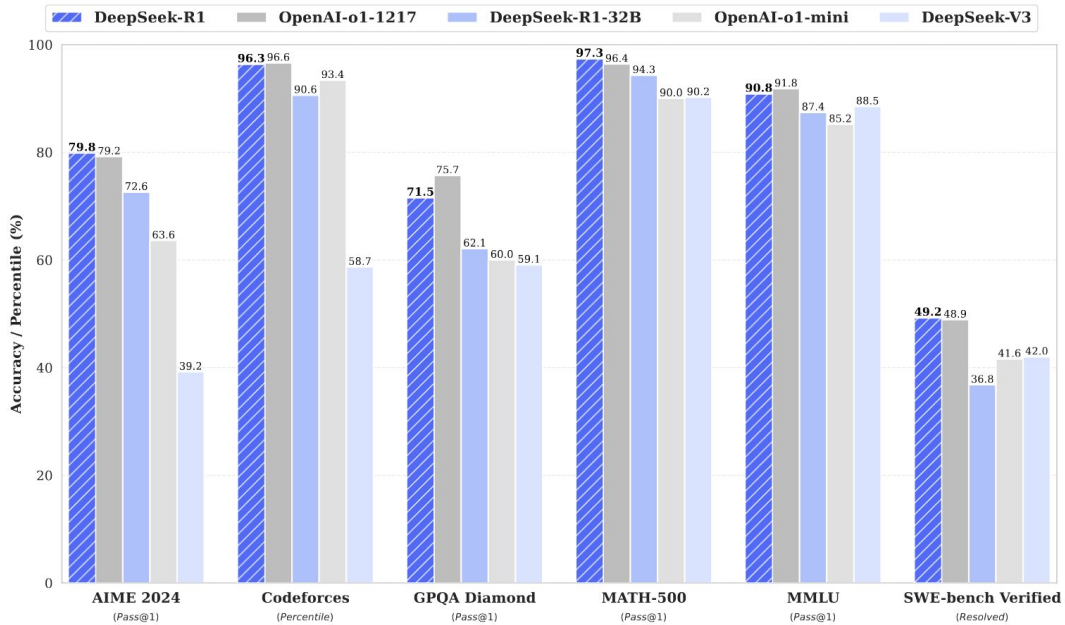
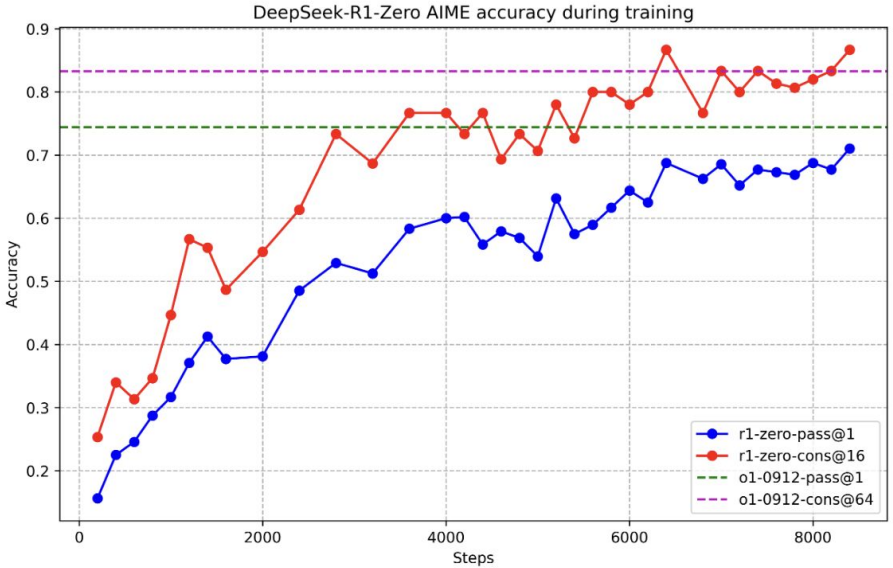
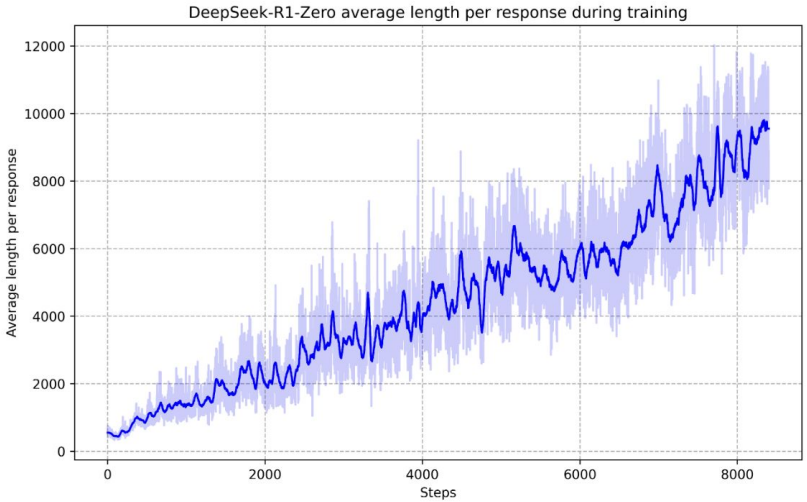


Figure 1 | Benchmark performance of DeepSeek-R1.

IMPERIAL

DeepSeek-R1





IMPERIAL

Part 1: DeepSeek-R1 Zero -> DeepSeek-R1

DeepSeek-R1 Zero -> DeepSeek-R1

DeepSeek-R1 Zero has impressive performance, however, lacks human readability. E.g. Multiple languages mixed together.

Method of improving upon DeepSeek-R1 Zero:

1. Create high-quality “cold-start” data with CoT for SFT.
2. RL after that.
3. 2nd SFT Phase for improved readability: (800K samples -> train Base model on this data).
    a. Reasoning Data. Sample via “rejection sampling” high quality CoT from RL model (incl. evaluation using Base Model; or readability params; like code blocks or multi-linguality).
    b. General Purpose data: Use DeepSeek-V3 (base model) SFT data for general purpose tasks.
4. RL tune (stage 3) for both Reasoning (R1-style) and general tasks (V3-style). **[question do they discard Stage 1 & 2?]**

Part 2: Interesting Insights

DeepSeek-R1: Interesting Insights

Distillation vs. RL

Model	AIME 2024		MATH-500	GPQA Diamond	LiveCodeBench
	pass@1	cons@64	pass@1	pass@1	pass@1
QwQ-32B-Preview	50.0	60.0	90.6	54.5	41.9
DeepSeek-R1-Zero-Qwen-32B	47.0	60.0	91.6	55.0	40.2
DeepSeek-R1-Distill-Qwen-32B	**72.6**	**83.3**	**94.3**	**62.1**	**57.2**

Table 6 | Comparison of distilled and RL Models on Reasoning-Related Benchmarks.

DeepSeek-R1: Interesting Insights

Unsuccessful Attempts:

Process Reward Model (PRM) PRM is a reasonable method to guide the model toward better approaches for solving reasoning tasks (Lightman et al., 2023; Uesato et al., 2022; Wang et al., 2023). However, in practice, PRM has three main limitations that may hinder its ultimate success. First, it is challenging to explicitly define a fine-grain step in general reasoning. Second, determining whether the current intermediate step is correct is a challenging task. Automated annotation using models may not yield satisfactory results, while manual annotation is not conducive to scaling up. Third, once a model-based PRM is introduced, it inevitably leads to reward hacking (Gao et al., 2022), and retraining the reward model needs additional training resources and it complicates the whole training pipeline. In conclusion, while PRM demonstrates a good ability to rerank the top-N responses generated by the model or assist in guided search (Snell et al., 2024), its advantages are limited compared to the additional computational overhead it introduces during the large-scale reinforcement learning process in our experiments.

Monte Carlo Tree Search (MCTS) Inspired by AlphaGo (Silver et al., 2017b) and AlphaZero (Silver et al., 2017a), we explored using Monte Carlo Tree Search (MCTS) to enhance test-time compute scalability. This approach involves breaking answers into smaller parts to allow the model to explore the solution space systematically. To facilitate this, we prompt the model to generate multiple tags that correspond to specific reasoning steps necessary for the search. For training, we first use collected prompts to find answers via MCTS guided by a pre-trained value model. Subsequently, we use the resulting question-answer pairs to train both the actor model and the value model, iteratively refining the process.

However, this approach encounters several challenges when scaling up the training. First, unlike chess, where the search space is relatively well-defined, token generation presents an exponentially larger search space. To address this, we set a maximum extension limit for each node, but this can lead to the model getting stuck in local optima. Second, the value model directly influences the quality of generation since it guides each step of the search process. Training a fine-grained value model is inherently difficult, which makes it challenging for the model to iteratively improve. While AlphaGo's core success relied on training a value model to progressively enhance its performance, this principle proves difficult to replicate in our setup due to the complexities of token generation.

In conclusion, while MCTS can improve performance during inference when paired with a pre-trained value model, iteratively boosting model performance through self-search remains a significant challenge.

DeepSeek-R1: Interesting Insights (Future Work)

- **General Capability:** Currently, the capabilities of DeepSeek-R1 fall short of DeepSeek-V3 in tasks such as function calling, multi-turn, complex role-playing, and JSON output. Moving forward, we plan to explore how long CoT can be leveraged to enhance tasks in these fields.
- **Language Mixing:** DeepSeek-R1 is currently optimized for Chinese and English, which may result in language mixing issues when handling queries in other languages. For instance, DeepSeek-R1 might use English for reasoning and responses, even if the query is in a language other than English or Chinese. We aim to address this limitation in future updates.
- **Prompting Engineering:** When evaluating DeepSeek-R1, we observe that it is sensitive to prompts. Few-shot prompting consistently degrades its performance. Therefore, we recommend users directly describe the problem and specify the output format using a zero-shot setting for optimal results.
- **Software Engineering Tasks:** Due to the long evaluation times, which impact the efficiency of the RL process, large-scale RL has not been applied extensively in software engineering tasks. As a result, DeepSeek-R1 has not demonstrated a huge improvement over DeepSeek-V3 on software engineering benchmarks. Future versions will address this by implementing rejection sampling on software engineering data or incorporating asynchronous evaluations during the RL process to improve efficiency.

Conclusion & Takeaways

Conclusions & Takeaways

1. Reading old-papers is very relevant (both for understanding & developing new methods, e.g. PPO) and for building new methods (e.g. fastText).
2. GRPO is quite a successful algorithm. In it's classical variant it seems to help the model distribution (i.e. Make TopK better).
3. The R1 paper introduces heuristic rewards & eliminates training PRM (process reward models), which seems to be very hard.
4. Distillation is more powerful than discovering knowledge by itself.
5. Pre-training data is crucial; data quality is crucial; data is crucial no matter what you do!
6. **Questions:** What can be elicited from a model directly? Vs. what needs to be distilled? What does this mean for completely new tasks (do we have a natural bottleneck, i.e. need humans first)?

Conclusions & Takeaways

Generally, this is an exciting avenue of research and still shows a lot of promise and uncharted territory.

Questions arise like: is the base model capable of these things and is RL bringing it out of the model? Or is RL doing fundamentally something new? How can one effectively combine and scale techniques together (V3 vs. R1)? How can one put everything together? How can one scale it to new problems?

References

Bibliography

- GRPO: https://arxiv.org/pdf/2402.03300
- DeepSeek-R1: https://arxiv.org/pdf/2501.12948
- PPO: https://arxiv.org/pdf/1707.06347
- MathShepard: https://arxiv.org/pdf/2312.08935
- Trust Region Policy Optimization (TRPO): https://arxiv.org/pdf/1502.05477 (that's the paper, where the p()/p_old estimate for the policy gradient comes from.
- Better Reasoning with Alignment: https://arxiv.org/pdf/2309.02144
- Weak to Strong Supervision (OpenAI): https://arxiv.org/pdf/2312.09390