DeepSeekMath

This paper not only introduces GRPO but shows how smaller LLMs can be pre-trained on an applied field from scratch, math in this paper; skip to section 4 for GRPO.
https : //github.com/deepseek − ai/DeepSeek − Math

When a section begins with *, it is a tangent about a mentioned topic and can be skipped.

Abstract

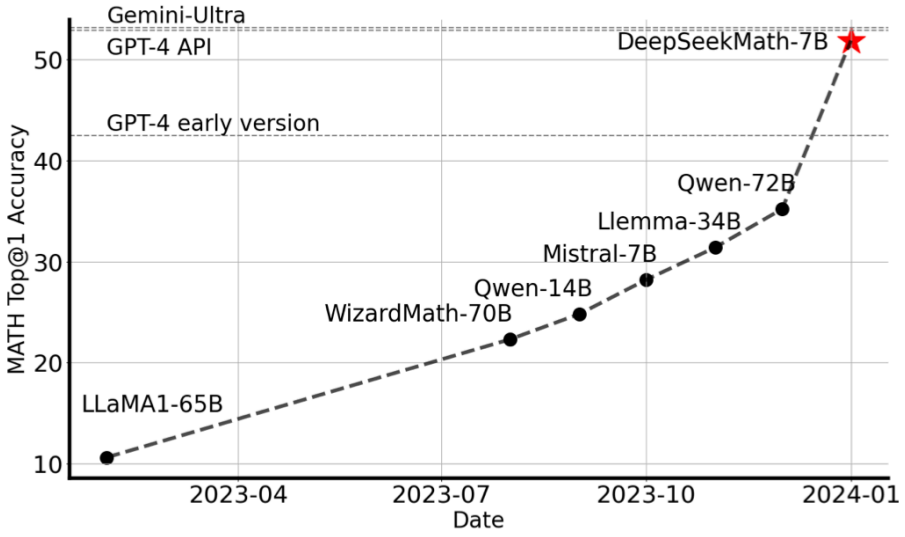
Problem Statement: Mathematical reasoning is a challenge for LLMs due to complex and structured nature.

Results:

- DeepSeekMath 7B achieves **51.7% on competition level MATH benchmark** without relying on external toolkits and voting techniques, approaching Gemini-Ultra & GPT4.
- With **self-consistency** over 64 samples, DeepSeekMath achieves **60.9% on MATH**.
  - Self-consistency is sampling from the model n times (64 in this case), and taking the most common answer (majority vote) as the final answer

Innovations:

- Meticulously engineered **data selection pipeline** from public web data
- **Group Relative Policy Optimization (GRPO)**, a variant of Proximal Policy Optimization (PPO), that enhances mathematical reasoning abilities while concurrently **optimizing the memory usage of PPO**



1. Introduction

LLMs have revolutionized the approach to mathematical reasoning in artificial intelligence. There have been significant advancements in both the quantitative reasoning benchmark and the geometry reasoning benchmark.

MATH Benchmark

The quantitive reasoning benchmark mentioned in this paper is the MATH benchmark, presented in the paper "Measuring Mathematical Problem Solving With the MATH Dataset"(Hendrycks et al., 2021). This is a dataset of 12,500 challenging competition math problems. There is a full step-by-step solution which can be used to teach models. At the time, the paper found that large transformers were not doing well at the benchmark. At the time, SOTA was GPT-3 at 5.2%. The benchmark is shown visually above, so, there has been significant progress with DeepSeekMath achieving 51.7% at only 7B parameters.

Additionally, these models have been proven instrumental in assisting humans in solving complex mathematical problems. Read this article by Terrance Tao for more on that ($https://unlocked.microsoft.com/ai-anthology/terence-tao/$). The issue is, all the best models are not publicly available, and open-source models are behind.

This paper aims to beat that:

- Create DeepSeek-Math Corpus, a large-scale high-quality pre-training corpus comprising 120B math tokens
  - Extracted from Common Crawl using a fastText-based classifier
  - Classifier is trained on OpenWebMath
  - After classifier, use human annotation to improve dataset quality

fastText Classifier

A fastText classifier is a lightweight and efficient text classification tool made by FAIR. It represents each word as a bag of character n-grams. For example, the word "hello" with n=3 would be broken into: <he, hel, ell, llo, lo>. This helps handle misspellings and rare words better. Unlike models that only look at whole words, fastText looks at parts of words; this is particularly useful for handling unknown words, technical terms, and words with common prefixes/suffixes. The architecture of fastText is that of a simple neural network. Words are converted to vectors (embeddings), those vectors are averaged to create a text representation, and the result is passed through a linear classifier. The main advantages of fastText in this use case (math scraping) is fast training and inference, good performance on specialized vocab (mathematical terms), ability to handle messy web text, and it is rather lightweight compared to larger LMs.

DeepSeekMath-Base is initialized with DeepSeek-Coder-Base-v1.5 7B, as the researchers notice starting from a code training model works better. After pre-training, they apply mathematical instruction tuning with chain-of-thought, program-of-thought, and tool integrated reasoning. After this, they use GRPO.

Contributions

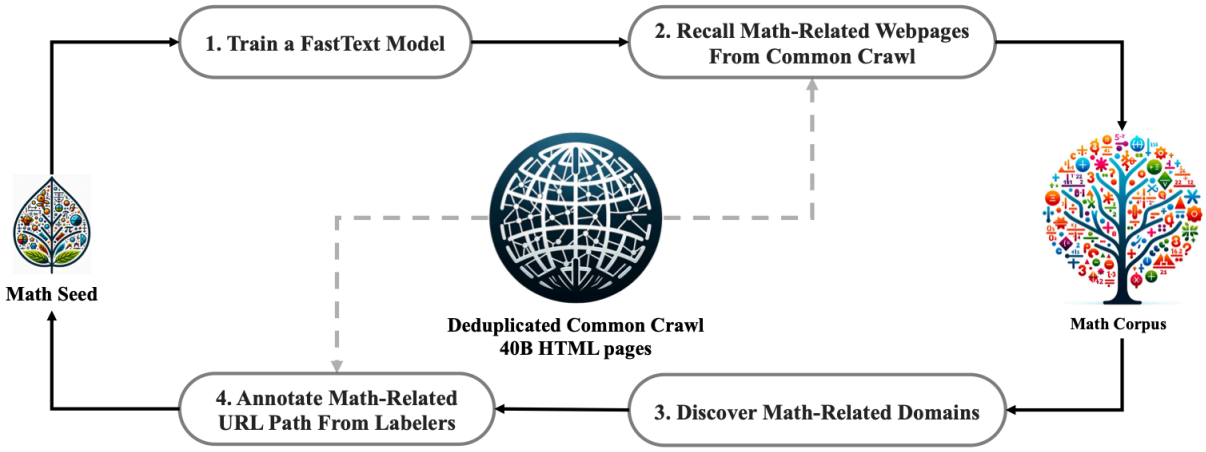
In this paper, they contribute:

- Math Pre-Training at Scale
- Exploration and Analysis of Reinforcement Learning

2. Math Pre-Training

2.1. Data Collection and Decontamination

The data collection process was an iterative pipeline. You start with OpenWebMath as seed data and trained a fastText classifier using positive (500,000 samples from OpenWebMath) and negative (500,000 random web pages from common crawl) examples. They used URL-based deduplication to reduce Common Crawl to 40B HTML pages. They then used fastText to identify and rank math pages.



The iterative improvement pipeline went like so:

1. After first round, identify math-focused domains
2. Manually annotate URLs with domains
3. Added uncollected content from the URLs to the seed corpus
4. Retrained the classifier and repeated the process
5. After 4 iterations, collected 35.5M math pages (120B toks)
6. Stopped at 4 iters

Decontamination process consisted of removing content matched benchmark test sets.

2.2 Training and Evals

They applied math training to a general pre-trained language model with 1.3B params, which shares the same architecture of DeepSeek LLMs. All experiments are conducted using HAI-LLM training framework.

DeepSeekMath corpus beats MathPile, OpenWebMath and Proof-Pile-2 on all metrics. The way they gaged this is by pretraining a LLM on each of these and comparing their performance.

Training Configuration: The bae model was DeepSeek-Coder-Base-v1.5 7B. The training duration was 500B tokens. The data distribution consisted of 56% DeepSeekMath corpus, 4% AlgebraicStack, 10% arXiv, 20% Github code, and 10% natural language (English/Chinese). The batch size was 10M tokens with a max learning rate of 4.2e-4.

Evaluation Areas: The model was evaluated on multiple areas: Mathematical Problem Solving with Step-by-Step Reasoning, Mathematical Problem Solving with Tool Use, Formal Mathematics and General Capabilities. The key achievement is showing strong performance across all areas and only at 7B params.

Mathematical Problem Solving with Step-by-Step Reasoning

Model	Size	GSM8K	MATH	OCW	SAT	MMLU STEM	CMATH	Gaokao MathCloze	Gaokao MathQA
Closed-Source Base Model									
Minerva	7B	16.2%	14.1%	7.7%	–	35.6%	–	–	–
Minerva	62B	52.4%	27.6%	12.0%	–	53.9%	–	–	–
Minerva	540B	58.8%	33.6%	17.6%	–	63.9%	–	–	–
Open-Source Base Model									
Mistral	7B	40.3%	14.3%	9.2%	71.9%	51.1%	44.9%	5.1%	23.4%
Llemma	7B	37.4%	18.1%	6.3%	59.4%	43.1%	43.4%	11.9%	23.6%
Llemma	34B	54.0%	25.3%	10.3%	71.9%	52.9%	56.1%	11.9%	26.2%
DeepSeekMath-Base	7B	64.2%	36.2%	15.4%	84.4%	56.5%	71.7%	20.3%	35.3%

Mathematical Problem Solving with Tool Use

Model	Size	GSM8K+Python	MATH+Python	miniF2F-valid	miniF2F-test
Mistral	7B	48.5%	18.2%	18.9%	18.0%
CodeLlama	7B	27.1%	17.2%	16.3%	17.6%
CodeLlama	34B	52.7%	23.5%	18.5%	18.0%
Llemma	7B	41.0%	18.6%	20.6%	22.1%
Llemma	34B	64.6%	26.3%	21.0%	21.3%
DeepSeekMath-Base	7B	66.9%	31.4%	25.8%	24.6%

Formal Mathematics

Model	Size	MMLU	BBH	HumanEval (Pass@1)	MBPP (Pass@1)
Mistral	7B	62.4%	55.7%	28.0%	41.4%
DeepSeek-Coder-Base-v1.5†	7B	42.9%	42.9%	40.2%	52.6%
DeepSeek-Coder-Base-v1.5	7B	49.1%	55.2%	43.2%	60.4%
DeepSeekMath-Base	7B	54.9%	59.5%	40.9%	52.6%

3. Supervised Fine-Tuning

3.1. SFT Data Curation

They constructed a mathematical instruction-tuning datasets with English and Chinese problems from different mathematical fields and of varying complexity levels. Problems are paired with chain-of-thought (COT), program-of-thought(PoT) and tool-integrated reasoning format. The total number of training examples is 776K.

3.2. Training and Evaluating DeepSeekMath-Instruct 7B

DeepSeekMath-Base + Instruction Tuning = DeepSeekMath-Instruct 7B. Training examples are randomly concatenated until reaching a max context length of 4K tokens. Model trained for 500 steps with batch size 256 and constant learning rate 5e-5. Check the paper for tables. The Instruct model surpasses all open-source models and the majority of proprietary models, specifically on step-by-step reasoning. This model excels at the MATH benchmark, and matches (SOTA) performance of DeepSeek-LLM-Chat 67B on the rest.

4. Reinforcement Learning

4.1. Group Relative Policy Optimization

RL has been proven to be effective in further improving the mathematical reasoning ability of LLMs after the SFT stage. This is why Group Relative Policy Optimization is introduced.

4.1.1. From PPO to GRPO

Proximal Policy Optimization (PPO) (Schulman et al., 2017) is an actor-critic RL algorithm that is widely used in the RL fine-tuning stage of LLMs.

Proximal Policy Optimization

PPO is a type of actor-critic reinforcement learning algorithm that improves training stability by avoiding excessively large policy updates. To do that, there is a ratio that indicates the difference between our current and old policy and clips this ratio from a specific range $[1-\varepsilon, 1+\varepsilon]$*. Doing this will ensure that the policy will not be large and that the training is more stable. The process of PPO is: sample experiences using the current policy, calculates the advantages (measurement of how much better/worse an action is compared to what was expected), updates the policy within the clipped bounds, and repeats.*

In particular, PPO optimizes LLMs by maximizing the following surrogate objective:

$$\mathrm{J}_{PPO}(\theta) = \mathrm{E}[q \sim P(Q), o \sim \pi_{\theta_{old}}(O|q)] \frac{1}{|o|} \sum_{t=1}^{|o|} \min \left[\frac{\pi_\theta(o_t|q,o_{<t})}{\pi_{\theta_{old}}(o_t|q,o_{<t})} A_t, \mathrm{clip}\left(\frac{\pi_\theta(o_t|q,o_{<t})}{\pi_{\theta_{old}}(o_t|q,o_{<t})}, 1-\varepsilon, 1+\varepsilon \right) A_t \right]$$

That equation looks quite complex, but it's essentially describing how to update a policy in a safe way. Let's break it down piece by piece for clarity:

1. $\mathrm{J}_{PPO}(\theta)$ is just the objective function we are trying to maximize where θ represents the parameters of the current policy.
2. $\mathrm{E}[q \sim P(Q), o \sim \pi_{\theta_{old}}(O|q)]$ represents taking the expected value over
   - Questions/prompts (q) sampled from some distribution P(Q)
   - Outputs (o) generated using our old policy $\pi_{\theta_{old}}$
3. $\frac{1}{|o|} \sum_{t=1}^{|o|}$ means we are averaging over all time steps in the output sequence.
4. The core part is the min operation between two terms
   - $\frac{\pi_\theta(o_t|q,o_{<t})}{\pi_{\theta_{old}}(o_t|q,o_{<t})}$ is the probability ratio (new policy probability / old policy probability)
   - A_t is the advantage (how much better/worse the action was compared to expected)
   - $\mathrm{clip}\left(\frac{\pi_\theta(o_t|q,o_{<t})}{\pi_{\theta_{old}}(o_t|q,o_{<t})}, 1-\varepsilon, 1+\varepsilon \right)$ keeps the ratio between $1-\varepsilon$ and $1+\varepsilon$ (typically $\varepsilon = 0.2$)

Think about it like putting guardrails on how much the AI's strategy can change in one update. It's like telling your model, "you can try new things, but don't go too crazy with changes all at once."

In PPO a value function needs to be trained alongside the policy model to mitigate over-optimization of the reward model, the standard approach is to add a per-token KL penalty from a reference model in the reward at each token, i.e.,

$$r_t = r_\phi(q, o_{\leq t}) - \beta \log \frac{\pi_\theta(o_t|q,o_{<t})}{\pi_{ref}(o_t|q,o_{<t})}$$

Let me break down this reward function as well. First, looking at the variables:

- r_t is the reward at time t (for each token)
- V_ϕ is a value function that predicts future rewards
- π_{ref} is a reference model (usually the SFT model)
- r_ϕ is the reward model

Looking at the equation, it can be broken down into two parts:

1. $r_\phi(q, o_{\leq t})$ is the base reward. This is what the reward model thinks about the quality of the output so far.

2. $\beta \log \frac{\pi_\theta(o_t|q,o_{<t})}{\pi_{ref}(o_t|q,o_{<t})}$ is the KL penalty. This measures how far the current model (π_θ) has drifted from the reference model. β controls how much the drift is penalized. It's sort of like a leash keeping the model from straying too far from it's original behavior.

Problem: The value function in PPO is typically another model of comparable size as the policy model. It brings a substantial memory and computational burden. Additionally, during RL training, the value function is treated as a baseline in the calculation of advantage for variance reduction. While in the LLM context, usually only the last token is assigned a reward score by the reward model, which may complicate the training of the value function that is accurate at each token.

Solution: DeepSeek proposed **Group Relative Policy Optimization**, which obviates the need for additional value function approximation as in PPO, and instead uses the average reward of multiple sampled outputs, produced in response to the same question, as the baseline.

More specifically, for each question q, GRPO samples a group of outputs $\{o_1, o_2, \ldots, o_G\}$ from the old policy $\pi_{\theta_{\text{old}}}$ and then optimizes the policy model by maximizing the following objective:

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

This GRPO equation is also quite complex, but it is essentially describing how to update a policy using group comparisons. Let's break it down piece by piece also:

1. $\mathrm{J}_{GRPO}(\theta)$ is our objective function we want to maximize, where θ represents our policy parameters, just like in the PPO equation.
2. $\mathrm{E}[q \sim P(Q), {o_i}_{i=1}^{G} \sim \pi_{\theta_{old}}(O|q)]$ represents taking the expected value over:
   - Question (q) sampled from distribution $P(Q)$
   - A group of G outputs (typically 64) generated using the old policy
   - This is a key difference from PPO - we generate multiple outputs for each question
3. $\frac{1}{G} \sum_{i=1}^{G} \frac{1}{|o_i|} \sum_{t=1}^{|o_i|}$ is:
   - First averaging all tokens in each output ($\frac{1}{|o_i|}$)
   - Then averaging over all G outputs in the group ($\frac{1}{G}$)
4. The core part of this equation has two main terms:
   - A policy ratio term similar to PPO, but using group-based advantages $\hat{A}_{i,t}$ instead of regular advantages
   - A KL divergence term $\beta \mathrm{D}_{KL}[\pi_\theta || \pi_{ref}]$ that directly measures distance from reference model

The key difference from PPO is in how advantages are calculated. Instead of using a value function to estimate advantages, GRPO uses group comparisons. $\hat{A}_{i,t}$ is calculated by comparing each output's reward to the group's average. This eliminates the need for the separate value network that PPO requires.

Now, taking a look at how the rewards work:

- For "outcome supervision", the reward is just normalized within the group $\hat{A}_{i,t} = \frac{r_i - \text{mean(r)}}{\text{std(r)}}$ where r_i is the reward for output i
- For "process supervision", rewards are given at each reasoning step:
  - Each step gets its own normalized reward within the group
  - The advantage for a token is sum of all future step rewards

It is sort of like grading on a curve - instead of trying to predict an absolute score (like PPO), GRPO just compares each answer to its peers in the same batch. This makes training more efficient while still maintaining good guardrails on policy updates.

We also use a different KL penalty term than used in PPO, the KL divergence is estimated with the following unbiased estimator:

$$\mathrm{D}_{KL}[\pi_\theta||\pi_{ref}] = \frac{\pi_{ref}(o_{i,t}|q,o_{i,<t})}{\pi_\theta(o_{i,t}|q,o_{i,<t})} - \log\frac{\pi_{ref}(o_{i,t}|q,o_{i,<t})}{\pi_\theta(o_{i,t}|q,o_{i,<t})} - 1$$

Algorithm 1: Iterative Group Relative Policy Optimization
Input: initial policy model $\pi_{\theta_{init}}$; reward models r_ϕ; task prompts D; hyperparameters ε, β, μ
1 : policy model $\pi_\theta \leftarrow \pi_{\theta_{init}}$
2 : for iteration $= 1, \ldots, I$ do
3 : reference model $\pi_{ref} \leftarrow \pi_\theta$
4 : for step $= 1, \ldots, M$ do
5 : Sample a batch D_b from D
6 : Update the old policy model $\pi_{\theta_{old}} \leftarrow \pi_\theta$
7 : Sample G outputs $\{o_i\}_{i=1}^{G} \sim \pi_{\theta_{old}}(\cdot|q)$ for each question $q \in \mathrm{D}_b$
8 : Compute rewards $\{r_i\}_{i=1}^{G}$ for each sampled output o_i by running r_ϕ
9 : Compute $\hat{A}_{i,t}$ for the t-th token of o_i through group relative advantage estimation.
10 : for GRPO iteration $= 1, \ldots, \mu$ do
11 : Update the policy model π_θ by maximizing the GRPO objective (Equation 21)
12 : Update r_ϕ through continuous training using a replay mechanism.
Output: π_θ

4.1.2. Outcome Supervision RL with GRPO

For each question q, a group of outputs $\{o_1, o_2, \ldots, o_G\}$ are sampled from the old policy model $\pi_{\theta_{\text{old}}}$. A reward model is used to score the outputs, yielding G rewards $\mathbf{r} = \{r_1, r_2, \ldots, r_G\}$ correspondingly. These rewards are then normalize by subtracting the group average and dividing by the group standard deviation. Outcome supervision provides the normalized reward at the end of each output o_i and sets the advantages of all tokens in the output as the normalized reward, and then optimizes the policy by maximizing the objective defined in the GRPO equation.

4.1.3. Process Supervision RL with GRPO

Instead of just giving one reward at the end, it provides rewards at each step of the reasoning process. For each question, generate G outputs(like 64 different answers). Each output has multiple reasoning steps and gets its own reward score. Normalize rewards in the group using: $\tilde{r}_i = \frac{r_i - \text{mean}(R)}{\text{std}(R)}$ then take the advantage for each token $(\hat{A}_{i,t}) =$ sum of all future step rewards.

4.1.4 Iterative RL with GRPO

This is needed because the reward model may get stale/outdated as training progresses. This works by generating answers using the current policy then using those answers to create new training data for the reward model. Keep 10% of old training data (replay mechanism) and update the reward model regularly. Finally set a new reference point for KL divergence.

4.2. Training and Evaluating DeepSeekMath-RL

The parameters and config info is in the paper. Omitted. But, DeepSeekMath-RL 7B attains accuracies of 88.2% and 51.7% on GSM8K and MATH, respectively, utilizing COT reasoning. This surpasses all models in the 7B - 70B range as well as the majority of closed source models.

DeepSeekMath-RL 7B is only trained on chain-of-thought format instruction tuning data of both benchmarks, starting from DeepSeekMath-Instruct 7B. Despite the constrained scope of the training data, it outperforms the instruct model on all evaluation metrics, RL WORKS!

Conclusion

I have omitted the discussion section, because I want to start on implementation. Feel free to read it in the paper, it has some valuable lessons learned from DeepSeek and some tables explaining results.