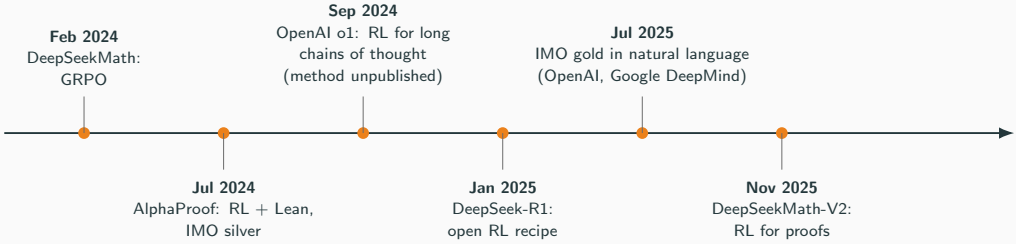
# Reinforcement learning for reasoning

Math 289

# Plan

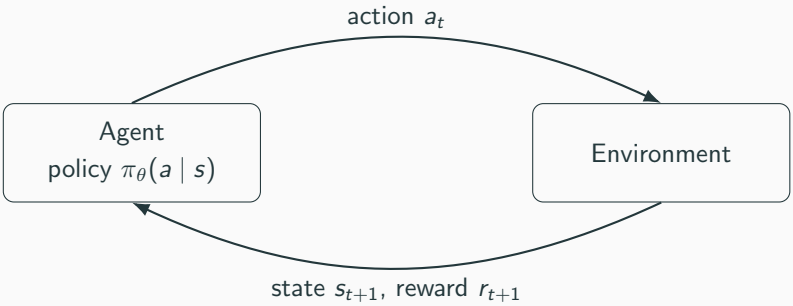
1. Timeline
2. Reinforcement Leaning (RL) basics and Group Relative Policy Optimization (GRPO)
3. Chain of thought reasoning
4. DeepSeek-R1
5. DeepSeekMath-V2: grading proofs
6. Multi-agent RL + Latent space reasoning

# Timeline


Feb 2024
DeepSeekMath:
GRPO
Jul 2024
AlphaProof: RL + Lean,
IMO silver
Sep 2024
OpenAI o1: RL for long
chains of thought
(method unpublished)
Jan 2025
DeepSeek-R1:
open RL recipe
Jul 2025
IMO gold in natural language
(OpenAI, Google DeepMind)
Nov 2025
DeepSeekMath-V2:
RL for proofs

# RL basics

# Reinforcement learning




Goal: choose $\theta$ to maximize expected total reward.

# RL for an LLM doing math

| RL | LLM |
|---|---|
| policy $\pi_\theta(a \mid s)$ | the LLM's next-token distribution |
| state $s_t$ | problem $q$ + tokens written so far $o_{<t}$ |
| action $a_t$ | next token $o_t$ |
| episode | one full response $o = (o_1, \ldots, o_T)$ |
| reward $r$ | 1 if the final answer is correct, else 0 (only at the end) |

$$\pi_\theta(o \mid q) = \prod_{t=1}^{T} \pi_\theta(o_t \mid q, o_{<t})$$

# The objective

Goal: Maximize expected reward

$$J(\theta) = \mathbb{E}_{q \sim \mathcal{D}} \, \mathbb{E}_{o \sim \pi_\theta(\cdot|q)} \big[ r(q, o) \big]$$

# The objective

Goal: Maximize expected reward

$$J(\theta) = \mathbb{E}_{q\sim\mathcal{D}}\, \mathbb{E}_{o\sim\pi_\theta(\cdot|q)}\big[r(q,o)\big]$$

Gradient ascent (or another optimizer):

$$\theta \leftarrow \theta + \eta\, \nabla_\theta J(\theta)$$

# The objective

Goal: Maximize expected reward

$$J(\theta) = \mathbb{E}_{q\sim\mathcal{D}}\, \mathbb{E}_{o\sim\pi_\theta(\cdot|q)}\big[r(q,o)\big]$$

Gradient ascent (or another optimizer):

$$\theta \leftarrow \theta + \eta\, \nabla_\theta J(\theta)$$

Fix q. How to estimate $\nabla_\theta J(\theta) = \sum_o \nabla \pi_\theta(o|q) r(q,o)$? Ideally compute samples and then average

# The objective

Goal: Maximize expected reward

$$J(\theta) = \mathbb{E}_{q\sim\mathcal{D}}\, \mathbb{E}_{o\sim\pi_\theta(\cdot|q)}\big[r(q,o)\big]$$

Gradient ascent (or another optimizer):

$$\theta \leftarrow \theta + \eta\, \nabla_\theta J(\theta)$$

Fix q. How to estimate $\nabla_\theta J(\theta) = \sum_o \nabla \pi_\theta(o|q) r(q,o)$? Ideally compute samples and then average

Issue: our samples come from the distribution $\pi_\theta(\cdot|q)$, but the sum above is not weighted by this

# Step 1: the log-derivative trick

Fix $q$. $\theta_0$: the parameters that generated the sample. Using $\nabla \pi = \pi \, \nabla \log \pi$:

$$\begin{aligned}
\nabla_\theta J(\theta_0) &= \sum_o \nabla_\theta \pi_\theta(o \mid q)\big|_{\theta_0} \, r(o) \\
&= \sum_o \pi_{\theta_0}(o \mid q) \, \nabla_\theta \log \pi_\theta(o \mid q)\big|_{\theta_0} \, r(o) \\
&= \mathbb{E}_{o \sim \pi_{\theta_0}} \left[ r(o) \, \nabla_\theta \log \pi_\theta(o \mid q)\big|_{\theta_0} \right]
\end{aligned}$$

# Step 1: the log-derivative trick

Fix $q$. $\theta_0$: the parameters that generated the sample. Using $\nabla \pi = \pi \, \nabla \log \pi$:

$$\begin{aligned}
\nabla_\theta J(\theta_0) &= \sum_o \nabla_\theta \pi_\theta(o \mid q)\big|_{\theta_0} \, r(o) \\
&= \sum_o \pi_{\theta_0}(o \mid q) \, \nabla_\theta \log \pi_\theta(o \mid q)\big|_{\theta_0} \, r(o) \\
&= \mathbb{E}_{o \sim \pi_{\theta_0}} \left[ r(o) \, \nabla_\theta \log \pi_\theta(o \mid q)\big|_{\theta_0} \right]
\end{aligned}$$

Side benefit: sums are more numerically stable than products

$$\nabla_\theta \log \pi_\theta(o \mid q) = \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(o_t \mid q, o_{<t})$$

# REINFORCE (Williams 1992)

Sample $o_1, \ldots, o_G \sim \pi_\theta(\cdot \mid q)$:

$$\hat{g} = \frac{1}{G} \sum_{i=1}^{G} r_i \, \nabla_\theta \log \pi_\theta(o_i \mid q) \qquad (\mathbb{E}\, \hat{g} = \nabla J)$$

# REINFORCE (Williams 1992)

Sample $o_1, \ldots, o_G \sim \pi_\theta(\cdot \mid q)$:

$$\hat{g} = \frac{1}{G} \sum_{i=1}^{G} r_i \, \nabla_\theta \log \pi_\theta(o_i \mid q) \qquad (\mathbb{E}\, \hat{g} = \nabla J)$$

$=$ gradient of

$$L(\theta) = \frac{1}{G} \sum_{i=1}^{G} r_i \, \log \pi_\theta(o_i \mid q)$$

- Every token of a correct response becomes more likely

# REINFORCE (Williams 1992)

Sample $o_1, \ldots, o_G \sim \pi_\theta(\cdot \mid q)$:

$$\hat{g} = \frac{1}{G} \sum_{i=1}^{G} r_i \, \nabla_\theta \log \pi_\theta(o_i \mid q) \qquad (\mathbb{E}\, \hat{g} = \nabla J)$$

$=$ gradient of

$$L(\theta) = \frac{1}{G} \sum_{i=1}^{G} r_i \, \log \pi_\theta(o_i \mid q)$$

- Every token of a correct response becomes more likely
- If $r \in \{0, 1\}$, easy problems waste gradient updates, and wrong answers aren't punished

# REINFORCE (Williams 1992)

Sample $o_1, \ldots, o_G \sim \pi_\theta(\cdot \mid q)$:

$$\hat{g} = \frac{1}{G} \sum_{i=1}^{G} r_i \, \nabla_\theta \log \pi_\theta(o_i \mid q) \qquad (\mathbb{E}\, \hat{g} = \nabla J)$$

$=$ gradient of

$$L(\theta) = \frac{1}{G} \sum_{i=1}^{G} r_i \, \log \pi_\theta(o_i \mid q)$$

- Every token of a correct response becomes more likely
- If $r \in \{0, 1\}$, easy problems waste gradient updates, and wrong answers aren't punished
- If one problem is repeated 10 times and only 1 of them is correct, the 1 success is most informative and its signal should be boosted

# Step 2: group-relative advantage

Sample a group of $G$ responses to the same $q$:

$$A_i = \frac{r_i - \text{mean}(r_1, \ldots, r_G)}{\text{std}(r_1, \ldots, r_G)} \qquad L(\theta) = \frac{1}{G} \sum_{i=1}^{G} A_i \log \pi_\theta(o_i \mid q)$$

# Step 2: group-relative advantage

Sample a group of $G$ responses to the same $q$:

$$A_i = \frac{r_i - \text{mean}(r_1, \ldots, r_G)}{\text{std}(r_1, \ldots, r_G)} \qquad L(\theta) = \frac{1}{G} \sum_{i=1}^{G} A_i \log \pi_\theta(o_i \mid q)$$

Learning comes from problems the model sometimes solves.

# Step 2: group-relative advantage

Sample a group of $G$ responses to the same $q$:

$$A_i = \frac{r_i - \text{mean}(r_1, \ldots, r_G)}{\text{std}(r_1, \ldots, r_G)} \qquad L(\theta) = \frac{1}{G} \sum_{i=1}^{G} A_i \log \pi_\theta(o_i \mid q)$$

Learning comes from problems the model sometimes solves.

**Next issue**: Our current setup alternates one sampling step with one gradient step (we assumed that the policy $\pi_\theta$ that generated the output $o$ is the same policy that gets a gradient update)

# Step 2: group-relative advantage

Sample a group of $G$ responses to the same $q$:

$$A_i = \frac{r_i - \text{mean}(r_1, \ldots, r_G)}{\text{std}(r_1, \ldots, r_G)} \qquad L(\theta) = \frac{1}{G} \sum_{i=1}^{G} A_i \log \pi_\theta(o_i \mid q)$$

Learning comes from problems the model sometimes solves.

**Next issue**: Our current setup alternates one sampling step with one gradient step (we assumed that the policy $\pi_\theta$ that generated the output $o$ is the same policy that gets a gradient update)

Sampling solutions is expensive and time consuming

# Step 2: group-relative advantage

Sample a group of $G$ responses to the same $q$:

$$A_i = \frac{r_i - \text{mean}(r_1, \ldots, r_G)}{\text{std}(r_1, \ldots, r_G)} \qquad L(\theta) = \frac{1}{G} \sum_{i=1}^{G} A_i \log \pi_\theta(o_i \mid q)$$

Learning comes from problems the model sometimes solves.

**Next issue**: Our current setup alternates one sampling step with one gradient step (we assumed that the policy $\pi_\theta$ that generated the output $o$ is the same policy that gets a gradient update)

Sampling solutions is expensive and time consuming

Ideally we would like to sample a batch of solutions, and then do multiple gradient updates

# Step 3: reusing samples

If output was sampled from $\pi_{\text{old}} = \pi_{\theta_{\text{old}}}$, then

# Step 3: reusing samples

If output was sampled from $\pi_{\text{old}} = \pi_{\theta_{\text{old}}}$, then

$$\mathbb{E}_{o\sim\pi_\theta}[A(o)] = \mathbb{E}_{o\sim\pi_{\text{old}}}\left[\frac{\pi_\theta(o)}{\pi_{\text{old}}(o)}\, A(o)\right]$$

# Step 3: reusing samples

If output was sampled from $\pi_{\text{old}} = \pi_{\theta_{\text{old}}}$, then

$$\mathbb{E}_{o\sim\pi_\theta}[A(o)] = \mathbb{E}_{o\sim\pi_{\text{old}}}\left[\frac{\pi_\theta(o)}{\pi_{\text{old}}(o)}\,A(o)\right]$$

$$L(\theta) = \frac{1}{G}\sum_{i=1}^{G}\rho_i(\theta)\,A_i, \qquad \rho_i(\theta) = \frac{\pi_\theta(o_i \mid q)}{\pi_{\text{old}}(o_i \mid q)}$$

# Step 3: reusing samples

If output was sampled from $\pi_{\text{old}} = \pi_{\theta_{\text{old}}}$, then

$$\mathbb{E}_{o\sim\pi_\theta}[A(o)] = \mathbb{E}_{o\sim\pi_{\text{old}}}\left[\frac{\pi_\theta(o)}{\pi_{\text{old}}(o)}\,A(o)\right]$$

$$L(\theta) = \frac{1}{G}\sum_{i=1}^{G}\rho_i(\theta)\,A_i, \qquad \rho_i(\theta) = \frac{\pi_\theta(o_i \mid q)}{\pi_{\text{old}}(o_i \mid q)}$$

At $\theta = \theta_{\text{old}}$, $\rho_i$'s gradients exactly match $\log \pi_\theta$ :

$$\nabla_\theta \rho_i(\theta)\Big|_{\theta=\theta_{\text{old}}} = \frac{\nabla_\theta \pi_\theta(o_i \mid q)\big|_{\theta=\theta_{\text{old}}}}{\pi_{\text{old}}(o_i \mid q)} = \nabla_\theta \log \pi_\theta(o_i \mid q)\Big|_{\theta=\theta_{\text{old}}}$$

## Step 3: reusing samples

If output was sampled from $\pi_{\text{old}} = \pi_{\theta_{\text{old}}}$, then

$$\mathbb{E}_{o\sim\pi_\theta}[A(o)] = \mathbb{E}_{o\sim\pi_{\text{old}}}\left[\frac{\pi_\theta(o)}{\pi_{\text{old}}(o)}\,A(o)\right]$$

$$L(\theta) = \frac{1}{G}\sum_{i=1}^{G}\rho_i(\theta)\,A_i, \qquad \rho_i(\theta) = \frac{\pi_\theta(o_i \mid q)}{\pi_{\text{old}}(o_i \mid q)}$$

At $\theta = \theta_{\text{old}}$, $\rho_i$'s gradients exactly match $\log \pi_\theta$ :

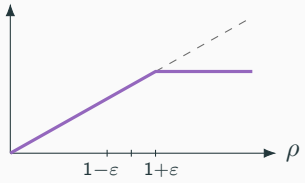
$$\nabla_\theta \rho_i(\theta)\Big|_{\theta=\theta_{\text{old}}} = \frac{\nabla_\theta \pi_\theta(o_i \mid q)\big|_{\theta=\theta_{\text{old}}}}{\pi_{\text{old}}(o_i \mid q)} = \nabla_\theta \log \pi_\theta(o_i \mid q)\Big|_{\theta=\theta_{\text{old}}}$$

**Next Issue:** After many steps $\pi_\theta$ drifts from $\pi_{\text{old}}$ and the estimate of the expectation gets worse, and the model can overfit to samples wher $\rho_i$ is really high.

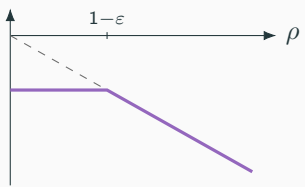
# Step 4: clipping (PPO, Schulman et al. 2017)

$$\min\left(\rho_i A_i,\ \text{clip}(\rho_i, 1-\varepsilon, 1+\varepsilon)\, A_i\right)$$

$A > 0$




$A < 0$



# GRPO (Shao et al. 2024)

$$L_{\mathrm{GRPO}}(\theta) = \mathbb{E}_{\substack{q\sim\mathcal{D} \\ \{o_i\}_{i=1}^{G}\sim\pi_{\mathrm{old}}(\cdot|q)}} \left[ \frac{1}{G}\sum_{i=1}^{G} \left( \min\left(\rho_i A_i,\ \mathrm{clip}(\rho_i, 1-\varepsilon, 1+\varepsilon) A_i\right) - \beta\, D_{\mathrm{KL}}(\pi_\theta \| \pi_{\mathrm{ref}}) \right) \right]$$

- $A_i$: compare to the group
- $\rho_i$: reuse samples
- clip: small steps make approximation more accurate
- KL: stay near $\pi_{\mathrm{ref}}$

DeepSeekMath v1 introduced GRPO and it became commonplace afterwards.

DeepSeekMath applies $\rho$ and the KL per token and averages over the tokens of $o_i$.

# Chain of thought

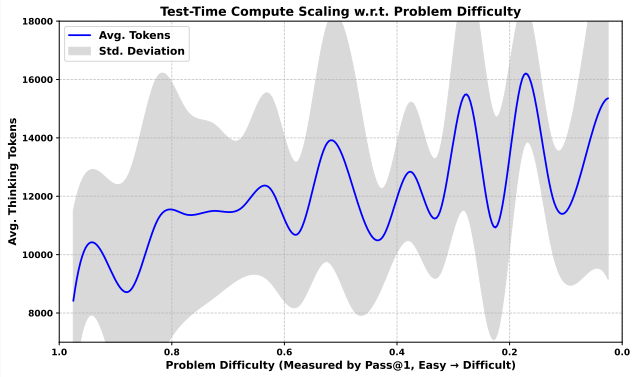
# Thinking blocks

Same model, same architecture, just put thinking in <think>...<think> blocks.

DeepSeek-R1-Zero training template:

A conversation between User and Assistant. The user asks a question, and the Assistant solves it. The assistant first thinks about the reasoning process in the mind and then provides the user with the answer. The reasoning process and answer are enclosed within <think>...</think> and <answer>...</answer> tags, respectively, i.e., <think> reasoning process here </think> <answer> answer here </answer>.
User: prompt. Assistant:

# More thinking on harder problems




DeepSeek-R1 on 2024 competition problems: under 7k thinking tokens on easy problems, over 18k on the hardest.

DeepSeek-AI, R1 paper (arXiv v2), Fig. 18

# DeepSeek-R1

# DeepSeek-R1 (Jan 2025; Nature, Sep 2025)

- Main contribution: RL on a pretrained model with reward only if final answer is correct $\Rightarrow$ long reasoning emerges

# DeepSeek-R1 (Jan 2025; Nature, Sep 2025)

- Main contribution: RL on a pretrained model with reward only if final answer is correct $\Rightarrow$ long reasoning emerges
- Base: DeepSeek-V3-Base MoE (671B parameters, 37B active per token)

# DeepSeek-R1 (Jan 2025; Nature, Sep 2025)

- Main contribution: RL on a pretrained model with reward only if final answer is correct $\Rightarrow$ long reasoning emerges
- Base: DeepSeek-V3-Base MoE (671B parameters, 37B active per token)
- RL cost $\approx$ \$294K (202K R1 Zero, 10K data creation, 82K R1) on top of the pretrained base

# DeepSeek-R1 (Jan 2025; Nature, Sep 2025)

- Main contribution: RL on a pretrained model with reward only if final answer is correct $\Rightarrow$ long reasoning emerges
- Base: DeepSeek-V3-Base MoE (671B parameters, 37B active per token)
- RL cost $\approx$ \$294K (202K R1 Zero, 10K data creation, 82K R1) on top of the pretrained base
- Fully open (training recipe, data, costs, weights) with performance matching OpenAI's closed o1 model

# R1-Zero: setup

- This is the main scientific contribution

# R1-Zero: setup

- This is the main scientific contribution
- Questions: 26k math, 25k code, 22k STEM, 15k logic (all auto-checkable)

# R1-Zero: setup

- This is the main scientific contribution
- Questions: 26k math, 25k code, 22k STEM, 15k logic (all auto-checkable)
- Reward $r = r_{\text{acc}} + r_{\text{format}}$
  - accuracy: boxed answer matches, or code passes tests
  - format: reasoning inside `<think>` tags

# R1-Zero: setup

- This is the main scientific contribution
- Questions: 26k math, 25k code, 22k STEM, 15k logic (all auto-checkable)
- Reward $r = r_{\text{acc}} + r_{\text{format}}$
  - accuracy: boxed answer matches, or code passes tests
  - format: reasoning inside `<think>` tags
- 10,400 training steps (gradient updates)

# R1-Zero: setup

- This is the main scientific contribution
- Questions: 26k math, 25k code, 22k STEM, 15k logic (all auto-checkable)
- Reward $r = r_{\text{acc}} + r_{\text{format}}$
  - accuracy: boxed answer matches, or code passes tests
  - format: reasoning inside `<think>` tags
- 10,400 training steps (gradient updates)
- Update $\pi_{old}$ every 16 steps, update $\pi_{ref}$ every 400 steps

# Training Loop

- Randomly take 512 questions from dataset

# Training Loop

- Randomly take 512 questions from dataset
- Sample 16 answers per question using $\pi_{old}$ (total of 8192 answers)

# Training Loop

- Randomly take 512 questions from dataset
- Sample 16 answers per question using $\pi_{old}$ (total of 8192 answers)
- Save each sampled token's probability under $\pi_{old}$ and $\pi_{ref}$

# Training Loop

- Randomly take 512 questions from dataset
- Sample 16 answers per question using $\pi_{old}$ (total of 8192 answers)
- Save each sampled token's probability under $\pi_{old}$ and $\pi_{ref}$
- Compute each answer's reward and relative advantage with the other answers of the same question

# Training Loop

- Randomly take 512 questions from dataset
- Sample 16 answers per question using $\pi_{old}$ (total of 8192 answers)
- Save each sampled token's probability under $\pi_{old}$ and $\pi_{ref}$
- Compute each answer's reward and relative advantage with the other answers of the same question
- Randomly split into batches of 32 questions (each with 16 answers, total of 512 Q-A pairs)

# Training Loop

- Randomly take 512 questions from dataset
- Sample 16 answers per question using $\pi_{old}$ (total of 8192 answers)
- Save each sampled token's probability under $\pi_{old}$ and $\pi_{ref}$
- Compute each answer's reward and relative advantage with the other answers of the same question
- Randomly split into batches of 32 questions (each with 16 answers, total of 512 Q-A pairs)

Gradient step:

- Run current $\pi_\theta$ on all 512 pairs in batch to get probability of each token

# Training Loop

- Randomly take 512 questions from dataset
- Sample 16 answers per question using $\pi_{old}$ (total of 8192 answers)
- Save each sampled token's probability under $\pi_{old}$ and $\pi_{ref}$
- Compute each answer's reward and relative advantage with the other answers of the same question
- Randomly split into batches of 32 questions (each with 16 answers, total of 512 Q-A pairs)

Gradient step:

- Run current $\pi_\theta$ on all 512 pairs in batch to get probability of each token
- Do one step of gradient ascent on GRPO objective (or with another optimizer)

# Training Loop

- Randomly take 512 questions from dataset
- Sample 16 answers per question using $\pi_{old}$ (total of 8192 answers)
- Save each sampled token's probability under $\pi_{old}$ and $\pi_{ref}$
- Compute each answer's reward and relative advantage with the other answers of the same question
- Randomly split into batches of 32 questions (each with 16 answers, total of 512 Q-A pairs)

Gradient step:

- Run current $\pi_\theta$ on all 512 pairs in batch to get probability of each token
- Do one step of gradient ascent on GRPO objective (or with another optimizer)
- Do the remaining batches of 512 Q-A pairs ($8192/512 = 16$ batches)

# Training Loop

- Randomly take 512 questions from dataset
- Sample 16 answers per question using $\pi_{old}$ (total of 8192 answers)
- Save each sampled token's probability under $\pi_{old}$ and $\pi_{ref}$
- Compute each answer's reward and relative advantage with the other answers of the same question
- Randomly split into batches of 32 questions (each with 16 answers, total of 512 Q-A pairs)

Gradient step:

- Run current $\pi_\theta$ on all 512 pairs in batch to get probability of each token
- Do one step of gradient ascent on GRPO objective (or with another optimizer)
- Do the remaining batches of 512 Q-A pairs ($8192/512 = 16$ batches)
- Update $\pi_{old}$

# Training Loop

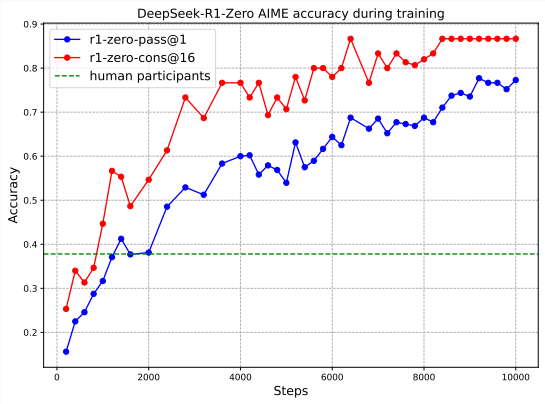
- Randomly take 512 questions from dataset
- Sample 16 answers per question using $\pi_{old}$ (total of 8192 answers)
- Save each sampled token's probability under $\pi_{old}$ and $\pi_{ref}$
- Compute each answer's reward and relative advantage with the other answers of the same question
- Randomly split into batches of 32 questions (each with 16 answers, total of 512 Q-A pairs)

Gradient step:

- Run current $\pi_\theta$ on all 512 pairs in batch to get probability of each token
- Do one step of gradient ascent on GRPO objective (or with another optimizer)
- Do the remaining batches of 512 Q-A pairs ($8192/512 = 16$ batches)
- Update $\pi_{old}$

Do this 1.6 times over all questions in dataset

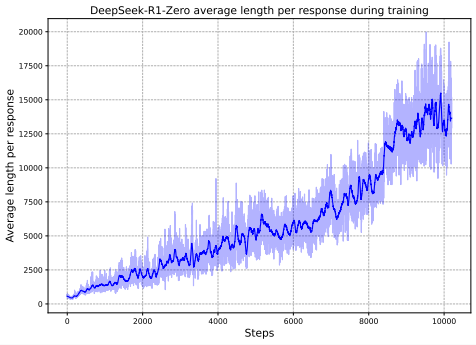
# Accuracy during RL



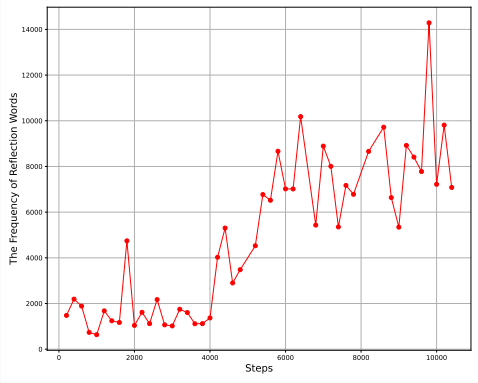

AIME 2024 pass@1: 15.6% $\rightarrow$ 77.9%  (majority of 16: 86.7%)

R1 paper, Fig. 1(a)

# Longer answers, more self-checking




average response length

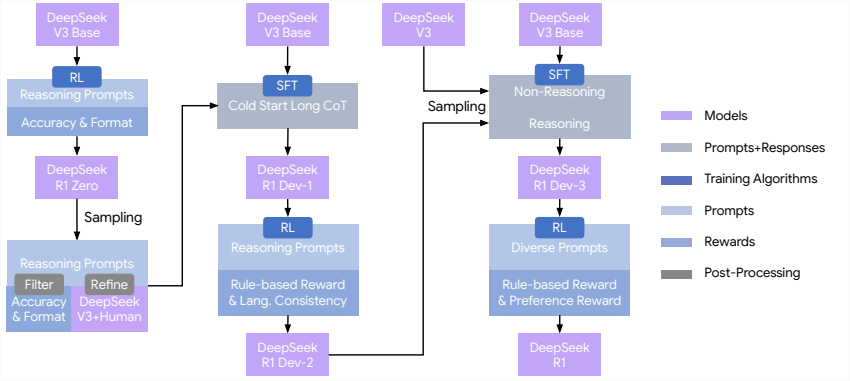



frequency of reflection words (“wait”, . . . )

R1 paper, Figs. 1(b) and 9

# From R1-Zero to R1

R1-Zero: hard to read, mixes English and Chinese, weak at writing.




R1 paper, Fig. 2

# What improved

| | V3 | R1-Zero | R1 |
|---|---|---|---|
| AIME 2024 (pass@1) | 39.2 | 77.9 | 79.8 |
| MATH-500 | 90.2 | 95.9 | 97.3 |
| CNMO 2024 | 43.2 | 88.1 | 78.8 |
| GPQA Diamond (science) | 59.1 | 75.8 | 71.5 |
| Codeforces rating | 1134 | 1444 | 2029 |
| AlpacaEval 2.0 (writing) | 70.0 | 24.7 | 87.6 |
| IF-Eval (instructions) | 86.1 | 46.6 | 83.3 |

V3 and R1 share the same pretrained base.

| AIME 2025, average score | /15 |
|---|---|
| human participants | 6.2 |
| DeepSeek-V3 | 3.3 |
| OpenAI o1 | 12.0 |
| DeepSeek-R1 | 11.3 |

R1 paper, Tables 12, 13

# What did not work

**Process reward models**

- Train the model on a reasoning process
- hard to label steps correct, humans do not scale

**Monte Carlo tree search**

- What Harmonic/AlphaProof do, but DeepSeek R1 tries it with NL reasoning steps

# DeepSeekMath-V2: RL for proofs

# The problem

- A correct final answer does not mean a correct proof

# The problem

- A correct final answer does not mean a correct proof
- A proof has no final answer to easily check

# The problem

- A correct final answer does not mean a correct proof
- A proof has no final answer to easily check

Options for the reward:

- formal proof (Lean): exact, but the problem must be formalized

# The problem

- A correct final answer does not mean a correct proof
- A proof has no final answer to easily check

Options for the reward:

- formal proof (Lean): exact, but the problem must be formalized
- LLM grader: natural language, but can be wrong or reward hacked

DeepSeekMath-V2 (Nov 2025): train an LLM grader to be reliable, then use it as the reward.

# The problem

- A correct final answer does not mean a correct proof
- A proof has no final answer to easily check

Options for the reward:

- formal proof (Lean): exact, but the problem must be formalized
- LLM grader: natural language, but can be wrong or reward hacked

DeepSeekMath-V2 (Nov 2025): train an LLM grader to be reliable, then use it as the reward.

Base model: DeepSeek-V3.2-Exp-Base, 685B parameter MoE model

# The verifier's rubric

The verifier LLM was given a rubric to grade proofs

| score | meaning |
|---|---|
| 1 | completely correct, all steps executed properly and clearly demonstrated |
| 0.5 | generally correct, some details omitted or minor errors |
| 0 | does not address the problem, fatal errors, or severe omissions |

DeepSeekMath-V2, Appendix A

# Verifier

$\pi_\varphi(\, V, s' \mid X, Y)$: reads problem $X$ and proof $Y$, writes analysis $V$ (issues found) and score $s'$.

# Verifier

$\pi_\varphi(\, V, s' \mid X, Y)$: reads problem $X$ and proof $Y$, writes analysis $V$ (issues found) and score $s'$.

- Data: olympiad problems with proofs from an earlier model with human expert scores $s$

# Verifier

$\pi_\varphi(\, V, s' \mid X, Y)$: reads problem $X$ and proof $Y$, writes analysis $V$ (issues found) and score $s'$.

- Data: olympiad problems with proofs from an earlier model with human expert scores $s$
- RL reward:

$$R_V = R_{\text{format}} \cdot \big(1 - |s' - s|\big) = R_{\text{format}} \cdot R_{\text{score}}$$

# Verifier

$\pi_\varphi(\, V, s' \mid X, Y)$: reads problem $X$ and proof $Y$, writes analysis $V$ (issues found) and score $s'$.

- Data: olympiad problems with proofs from an earlier model with human expert scores $s$
- RL reward:

$$R_V = R_{\text{format}} \cdot \big(1 - |s' - s|\big) = R_{\text{format}} \cdot R_{\text{score}}$$

Issue: on a flawed proof the verifier can get $s'$ right while inventing issues.

# Meta-verifier

$\pi_\eta$: grades the *analysis* $V$. Are the issues real? Do they justify $s'$? Score $\in \{0, 0.5, 1\}$.

- trained by RL on expert ratings of analyses

New verifier reward:

$$R_V = R_{\text{format}} \cdot R_{\text{score}} \cdot R_{\text{meta}}$$

# Meta-verifier

$\pi_\eta$: grades the *analysis* $V$. Are the issues real? Do they justify $s'$? Score $\in \{0, 0.5, 1\}$.

- trained by RL on expert ratings of analyses

New verifier reward:

$$R_V = R_{\text{format}} \cdot R_{\text{score}} \cdot R_{\text{meta}}$$

Mechanically, the meta-verifier only checks statements that the verifier claimed were wrong. It does not check check things that the verifiers said was correct.

# Training pipeline: setup

1. Problems $D_p$: 17,503 proof problems crawled from AoPS

# Training pipeline: setup

1. Problems $D_p$: 17,503 proof problems crawled from AoPS
2. Proof labels $D_v$: a variant of DeepSeek-V3.2-Exp-Thinking writes proofs, then experts score a sample with 0, 0.5, or 1

# Training pipeline: setup

1. Problems $D_p$: 17,503 proof problems crawled from AoPS
2. Proof labels $D_v$: a variant of DeepSeek-V3.2-Exp-Thinking writes proofs, then experts score a sample with 0, 0.5, or 1
3. Train first verifier $\pi_\varphi$: start from a version of DeepSeek-V3.2-Exp-SFT and run GRPO on $D_v$ with reward $R_{\text{format}} \cdot R_{\text{score}}$.

# Training pipeline: setup

1. Problems $D_p$: 17,503 proof problems crawled from AoPS
2. Proof labels $D_v$: a variant of DeepSeek-V3.2-Exp-Thinking writes proofs, then experts score a sample with 0, 0.5, or 1
3. Train first verifier $\pi_\varphi$: start from a version of DeepSeek-V3.2-Exp-SFT and run GRPO on $D_v$ with reward $R_{\text{format}} \cdot R_{\text{score}}$.
4. Train meta-verifier $\pi_\eta$: experts score $\pi_\varphi$'s analyses, creating dataset $D_{mv}$. Then train $\pi_\eta$ by RL on those scores

# Training pipeline: repeating cycle

5. Improve the verifier and train it to meta-verify: train on $D_v$ with $R_{\text{format}} \cdot R_{\text{score}} \cdot R_{\text{meta}}$ ($R_{\text{meta}}$ comes from $\pi_\eta$) and then on $D_{mv}$ with $R_{\text{format}} \cdot R_{\text{score}}$. One model now does both

# Training pipeline: repeating cycle

5. Improve the verifier and train it to meta-verify: train on $D_v$ with $R_{\text{format}} \cdot R_{\text{score}} \cdot R_{\text{meta}}$ ($R_{\text{meta}}$ comes from $\pi_\eta$) and then on $D_{mv}$ with $R_{\text{format}} \cdot R_{\text{score}}$. One model now does both
6. Train generator $\pi_\theta$: start from the verifier checkpoint and run GRPO on $D_p$. **No proof correctness labels needed**. Proof generator writes a proof $Y$, then a self-analysis $Z$ with self-score of its own proof $s'$. The verifier scores then $Y$ as $s$ and meta-verifies $Z$:

$$R = R_{\text{format}}(Y, Z) \cdot \big(0.76\, R_Y + 0.24\, R_Z\big), \quad R_Y = s, \quad R_Z = R_{\text{score}}(s', s) \cdot R_{\text{meta}}(Z)$$

# Training pipeline: repeating cycle

5. Improve the verifier and train it to meta-verify: train on $D_v$ with $R_{\text{format}} \cdot R_{\text{score}} \cdot R_{\text{meta}}$ ($R_{\text{meta}}$ comes from $\pi_\eta$) and then on $D_{mv}$ with $R_{\text{format}} \cdot R_{\text{score}}$. One model now does both
6. Train generator $\pi_\theta$: start from the verifier checkpoint and run GRPO on $D_p$. **No proof correctness labels needed**. Proof generator writes a proof $Y$, then a self-analysis $Z$ with self-score of its own proof $s'$. The verifier scores then $Y$ as $s$ and meta-verifies $Z$:

$$R = R_{\text{format}}(Y, Z) \cdot \big(0.76\, R_Y + 0.24\, R_Z\big), \quad R_Y = s, \quad R_Z = R_{\text{score}}(s', s) \cdot R_{\text{meta}}(Z)$$

7. Label the generator's new proofs to train the verifier. Early on experts label them with verifier analyses as help. Later on just rely on majority vote of the verifier/meta-verifier

# Training pipeline: repeating cycle

5. Improve the verifier and train it to meta-verify: train on $D_v$ with $R_{\text{format}} \cdot R_{\text{score}} \cdot R_{\text{meta}}$ ($R_{\text{meta}}$ comes from $\pi_\eta$) and then on $D_{mv}$ with $R_{\text{format}} \cdot R_{\text{score}}$. One model now does both
6. Train generator $\pi_\theta$: start from the verifier checkpoint and run GRPO on $D_p$. **No proof correctness labels needed**. Proof generator writes a proof $Y$, then a self-analysis $Z$ with self-score of its own proof $s'$. The verifier scores then $Y$ as $s$ and meta-verifies $Z$:

$$R = R_{\text{format}}(Y, Z) \cdot \big(0.76\, R_Y + 0.24\, R_Z\big), \quad R_Y = s, \quad R_Z = R_{\text{score}}(s', s) \cdot R_{\text{meta}}(Z)$$

7. Label the generator's new proofs to train the verifier. Early on experts label them with verifier analyses as help. Later on just rely on majority vote of the verifier/meta-verifier
8. Merge verifier and generator into one model: rejection fine-tuning (keep positive examples and then do SFT) combines verification and generation skills into one checkpoint, which starts the next verifier

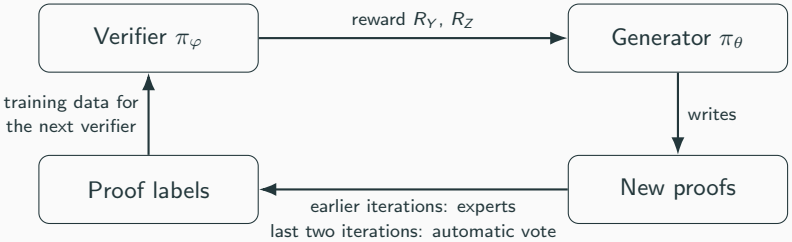
# Training pipeline: repeating cycle

5. Improve the verifier and train it to meta-verify: train on $D_v$ with $R_{\text{format}} \cdot R_{\text{score}} \cdot R_{\text{meta}}$ ($R_{\text{meta}}$ comes from $\pi_\eta$) and then on $D_{mv}$ with $R_{\text{format}} \cdot R_{\text{score}}$. One model now does both
6. Train generator $\pi_\theta$: start from the verifier checkpoint and run GRPO on $D_p$. **No proof correctness labels needed**. Proof generator writes a proof $Y$, then a self-analysis $Z$ with self-score of its own proof $s'$. The verifier scores then $Y$ as $s$ and meta-verifies $Z$:

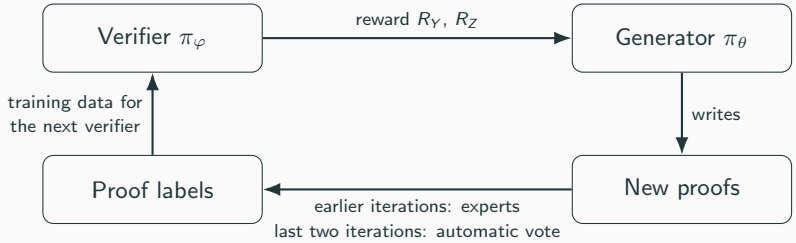
$$R = R_{\text{format}}(Y, Z) \cdot \big(0.76\, R_Y + 0.24\, R_Z\big), \quad R_Y = s, \quad R_Z = R_{\text{score}}(s', s) \cdot R_{\text{meta}}(Z)$$

7. Label the generator's new proofs to train the verifier. Early on experts label them with verifier analyses as help. Later on just rely on majority vote of the verifier/meta-verifier
8. Merge verifier and generator into one model: rejection fine-tuning (keep positive examples and then do SFT) combines verification and generation skills into one checkpoint, which starts the next verifier

# Training loop



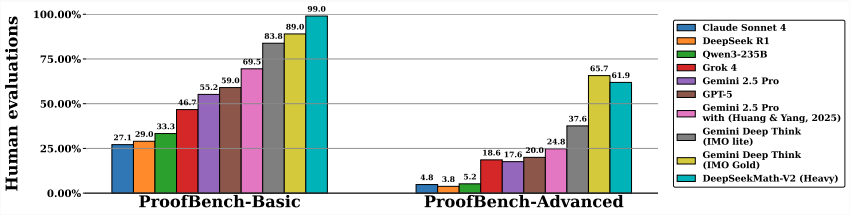
# Training loop




Automatic vote: $n$ verifier analyses per proof, $m$ meta-checks each. Of the verifier analyses that scored the lowest, if $\geq k$ meta-analyses says these are valid verifier analyses, then assign this lowest score. If no analyses find real issues, label 1. Otherwise, discard the proof or send it for human expert review

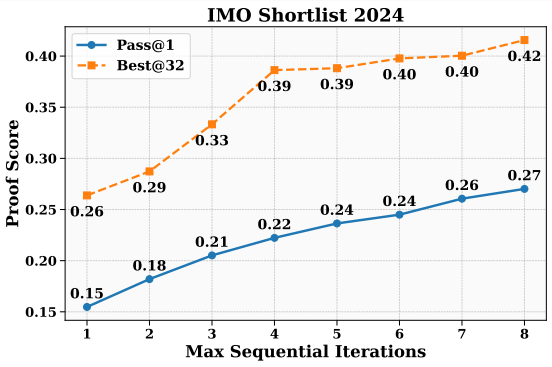
# Results

- IMO 2025: 5 of 6 problems (note DeepSeek paper is from November 2025)
- CMO 2024: gold
- Putnam 2024: 118/120




DeepSeekMath-V2, Fig. 3. With scaled test-time compute: up to 16 rounds of self-verified refinement.

# Self-verified refinement




Iterations of write then self-analyze then revise

DeepSeekMath-V2, Fig. 2

# Multi-agent RL

# Multi-agent RL

How can you get multiple agents to coordinate on one common task?

Ideally:

- Good information exchange between agents
- Splitting task into individual components
- Specialization in agent behavior

# Example

OpenAI Sol agents on a math task, given a common message board:

From `Agent011ExplicitFormulaPlanRequest.md`:

a001 thanks realL1bound majorprogress! Concrete nextfoundationaltask ifavailable: investigate smoothedexplicitformula via finitecontour residues/localzero-poleexpansion, without asserting estimates. ... Please coordinatea022 notduplicate genericcontourshift.

Proposing a mathematical approach. From `Agent014PhaseFreeFejerIdea_a019.md`:

IMPORTANT POSSIBLE PHASE-FREE LMO FIX (algebra checked mentally, not formal yet): no need rightmost positiveunitnode if use NONNEGATIVE MODULATION coefficients!

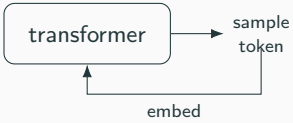
???:

c10designated c58→c84 DONE immutable95000 mathpool. a008 repliedc8leadrequiresconcretelogfree detectorplan; no currentcommit. [...] c9helpersfastsaving; itsproofclaimsifnotby1862missespossibleinactivity2462butunknownharnesssemantics.

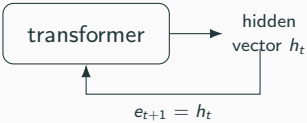
# Latent reasoning

# Idea

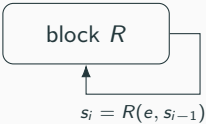
Chain of thought




Continuous thought



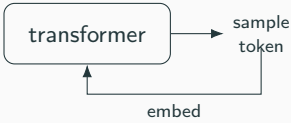

Looped layers



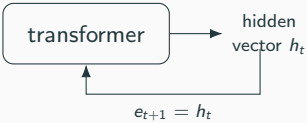

Reason in hidden states instead of written tokens.

# Idea

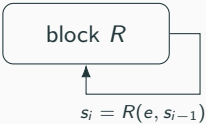
Chain of thought




Continuous thought




Looped layers

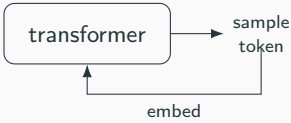



Reason in hidden states instead of written tokens.

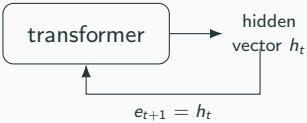
- Don't need to waste compute on "connective tissue" tokens

# Idea

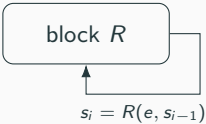
Chain of thought




Continuous thought




Looped layers

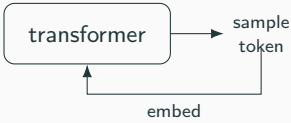



Reason in hidden states instead of written tokens.

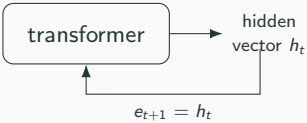
- Don't need to waste compute on "connective tissue" tokens
- Can have superposition of reasoning steps

# Idea

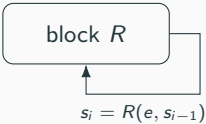
Chain of thought




Continuous thought




Looped layers

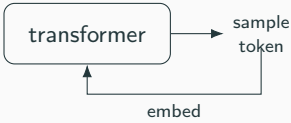



Reason in hidden states instead of written tokens.

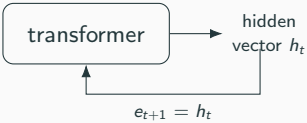
- Don't need to waste compute on "connective tissue" tokens
- Can have superposition of reasoning steps
- nothing human-readable is written

# Idea

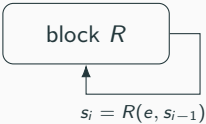
Chain of thought




Continuous thought




Looped layers




Reason in hidden states instead of written tokens.

- Don't need to waste compute on "connective tissue" tokens
- Can have superposition of reasoning steps
- nothing human-readable is written

Reference: Coconut, Hao et al 2024