LLM Reasoning

Lecture 3: LLM Reasoning via Reinforcement Learning

Yaya Sy

16/10/2025

Recap on repeated scaling

Scaling repeated sampling

- Scaling the ***number of attempts improves the performance*** of LLMs
- The performance as a function of **number of attempts *k* is determined by a power function**: $c = exp(ak^b)$
- L**arge models improve more quickly** when scaling the number of attempts

Reward Modeling

- For **verifiable tasks** (math, code, instructions), the reward is based on the **correctness** of the output
- For **non-verifiable tasks**:
    - **Self-confidence** can be a good signal (see lab on translation)
    - **LLM as a Judge** can be a good rewarder but they can be biased (prefer long answers)
    - One common solution is to **train a reward model on a preference dataset**
    - **Bradley-Terry** is a popular approach for training reward models.

Repeated Sampling is *time-consuming*

Repeated Sampling Objective: Sampling multiple outputs and pick the one with the highest reward score.

$$\boxed{\hat{y} = \arg\max_{y} \; R(x, y), \qquad y \sim p_\theta(\cdot \mid x)}$$

R(y) or reward(y) is the function that assigns higher ***score*** to better answers.

Problem: This takes some time and the user frustrates (people want fast answers!)

Solution: Training the model to directly maximize the reward, so no need for repeated sampling at inference-time

This is ***Reinforcement Learning***

Reinforcement Learning (RL)

Reinforcement Learning is learning to maximize a reward

In ***Repeated Sampling***, we ***search for the answer*** that maximizes the reward *R:*

$$\hat{y} = \arg\max_{y} \; R(x, y), \qquad \text{where } y \sim p(\cdot \mid x, \theta)$$

In ***Reinforcement Learning***, we ***search for the model*** that maximizes the reward *R:*

$$\hat{\theta} = \arg\max_{\theta} \; R(x, y), \qquad \text{where } y \sim p(\cdot \mid x, \theta)$$

We want a model that can generate answer with high reward.

Concepts and notations

- ***Policy***

$$\pi(\cdot) = p(\cdot) = LLM(\cdot)$$

- ***Expectation:*** You can think *average*

$$\hat{\theta} = \arg\max_{\theta} \frac{1}{N} \sum_{i=1}^{N} R(x, y_i), \qquad \text{where } y_i \sim \pi_\theta(\cdot \mid x)$$

$$\Big\updownarrow$$

$$\hat{\theta} = \arg\max_{\theta} \mathbb{E}_{y \sim \pi_\theta(\cdot|x)}[R(x, y)]$$

Reinforcement Learning is learning to maximize a reward

Let's say we ask to the LLM a question: *x* = *"you prefer cats or dogs?"*

The model answers $y \sim \pi_\theta(\cdot \mid x)$, and we reward it 10 if the answer is *cats* and 2 if it's *dogs*

What is the expected reward $\mathbb{E}_{y \sim \pi_\theta(\cdot|x)}[R(x,y)]$?

	y	$\pi_\theta(y \mid x)$	$R(x,y)$
Answer 1	"cats"	0.8	10
Answer 2	"dogs"	0.2	2

The expected reward in this specific example:

$$\mathbb{E}_{y \sim \pi_\theta(\cdot|x)}[R(x,y)] = 0.8 \times 10 + 0.2 \times 2 \ = 8.4$$

The expected reward in general:

$$\mathbb{E}_{y \sim \pi_\theta(\cdot|x)}[R(x,y)] = \sum_y \pi_\theta(y|x) R(x,y)$$

Reinforcement Learning is learning to maximize a reward

The expected reward in general:

$$\mathbb{E}_{y\sim\pi_\theta(\cdot|x)}[R(x,y)] = \sum_y \pi_\theta(y|x)R(x,y)$$

And we want to maximize this. To do so, we can use *gradient descent / ascent,* and for that we need to find the gradient:

$$\nabla\mathbb{E}_{y\sim\pi_\theta(\cdot|x)}[R(x,y)] = \nabla\sum_y \pi_\theta(y|x)R(x,y) = \sum_y R(x,y)\nabla\pi_\theta(y|x)$$

The problem of having the gradient wrt π_θ :

- The sampling method (top-p, etc) depends on the policy parameters but is not differentiable
- The reward in itself is often not differentiable (can be a human) and independent to the policy parameters

We need to find an estimation of the gradient of the policy

The REINFORCE approach

REINFORCE is one way to estimate the gradient of the policy:

- Ignores about the gradient of the sampling methods (top-p, top-k, etc.)
- Treats the reward as a constant
- Compute the log-probs of the answer, weighted by the reward. Optimize this directly.

In code:

```
x = “you prefer cats or dogs?”
for epoch in epochs:
     y = llm.generate(x)

     r = 10 if y == “cats” else 2

     logprobs = llm(y).logprobs() # compute log π(y|x)
     objective = (r * logprobs).mean()

     # maximizing an objective is minimizing its negative
     loss = -objective
```

In math:

$$\nabla \mathbb{E}_{y\sim\pi_\theta(\cdot|x)}[R(x,y)] = \mathbb{E}_{y\sim\pi_\theta(\cdot|x)}[R(x,y)\nabla \log \pi_\theta(y \mid x)]$$

The REINFORCE approach

→ REINFORCE is ***on-policy:***

- At each step, the model generates data.
- This fresh generated data is used to update the LLM in itself

The limits of on-policy reinforcement learning

→ In on-policy, the model generates fresh data at each training step

→ This is costly. Specially in the case of LLMs, where generation is costly.

→ What if the model generates data and do multiple training steps on it?

→ This is off-policy reinforcement learning

Off-policy reinforcement learning

In off-policy, we want to perform multiple training steps on the data generated by the model

Naively, it looks like that:

```
x = “you prefer cats or dogs?”
for epoch in epochs:
    y = llm.generate(x)
    r = 10 if y == “cats” else 2

    for step in off_policy_steps:
        logprobs = llm(y).logprobs()
        objective = (r * logprobs).mean()
        loss = -objective
        loss.backward()
        llm.update()
```

- At the first step, the algorithm is still on-policy
- But after the first step, the model is updated and is being trained on data generated by a *different* version. ***This is off-policy.***

Off-policy: a clash of generation

→ In off-policy, the model is being trained on data from a different version of the model. Since the data is from an old policy, the current model is optimizing the *wrong* expected reward. *Let's take the example "you prefer cats or dogs"*:

	y	$\pi_\theta(y \mid x)$	$R(x, y)$
Answer 1	"cats"	0.2	10
Answer 2	"dogs"	0.8	2

Old policy

Wants more dogs!

	y	$\pi_\theta(y \mid x)$	$R(x, y)$
Answer 1	"cats"	0.8	10
Answer 2	"dogs"	0.2	2

New policy

Wants more cats!

I want more dogs! I want more cats.

Old policy *New policy*

Since the data is from the old policy, the expected reward we're *actually* optimizing: 0.2 * 10 + 0.8 * 2 = 3.6

The expected reward we *want* to optimize (new policy): 0.8 * 10 + 0.2 * 2 = 8.4

There is a *mismatch* between what we are *actually* optimizing and what we *want* to optimize (the expected reward of the new policy).

Off-policy: a clash of generation

To fix this mismatch between old and new policy, we use ***importance ratio.***

Importance ratio takes into account the preferences of the new policy by down weighting or up weighting the scores

Importance ratio: $\boldsymbol{\pi}_\theta$ / $\boldsymbol{\pi}_{\text{old}}$

	y	$\pi_\theta(y \mid x)$	$\pi_{\text{old}}(y \mid x)$	$\frac{\pi_\theta(y \mid x)}{\pi_{\text{old}}(y \mid x)}$	$R(x, y)$
Answer 1	"cats"	0.8	**0.2**	**4**	10
Answer 2	"dogs"	0.2	**0.8**	**0.25**	2

Expected reward we're optimizing with importance ratio:
(0.2 * 4 * 10) + (0.8 * 0.25 * 2) = ***8.4*** ***(exactly what we want)***

	y	$\pi_\theta(y \mid x)$	$R(x, y)$
Answer 1	"cats"	0.2	10
Answer 2	"dogs"	0.8	2

Old policy ($\boldsymbol{\pi}_{\text{old}}$*)*

Expected reward we're optimizing:
0.2 * 10 + 0.8 * 2 = ***3.6***

	y	$\pi_\theta(y \mid x)$	$R(x, y)$
Answer 1	"cats"	0.8	10
Answer 2	"dogs"	0.2	2

New policy ($\boldsymbol{\pi}_\theta$*)*

Expected reward we *want* to optimize: 0.8 * 10 + 0.2 * 2 = ***8.4***

Off-policy with importance ratio

In code:

```
x = “you prefer cats or dogs?”
for epoch in epochs:
    y = llm.generate(x)
    r = 10 if y == “cats” else 2

    with torch.no_grad():
        old_logprobs = llm(y).logprobs()

    for step in off_policy_steps:
        new_logprobs = llm(y).logprobs()
        # log (a / b) = log a - log b:
        ratio = new_logprobs - old_logprobs
        objective = (r * ratio).mean()
        loss = -objective
        loss.backward()
        llm.update()
```

In math:

$$\mathbb{E}_{y\sim\pi_{\text{old}}(\cdot|x)}\left[\frac{\pi_\theta(y\mid x)}{\pi_{\text{old}}(y\mid x)}R(x,y)\right]$$

We are never enough careful…

If the new policy diverges too much from the old policy, the importance ratio can be ***extremely big*** or ***extremely small***

This leads to an ***excessively large policy updates***, creating instabilities.

$\pi_\theta / \pi_{\text{old}}$ is 1 when the old and new policy are the same. ***We would like to be close to 1***, with some ϵ-tolerance, to avoid large policy updates

We define 1+ϵ and 1-ϵ as ***the trust region*** and ***restrict (clip)*** the importance ratio there.

$$\text{unclipped} = \frac{\pi_\theta(y \mid x)}{\pi_{\text{old}}(y \mid x)} R(x, y)$$

$$\text{clipped} = \text{clip}\left(\frac{\pi_\theta(y \mid x)}{\pi_{\text{old}}(y \mid x)}\ 1 - \epsilon,\ 1 + \epsilon \right) R(x, y)$$

We can be even more careful by taking the minimum of the clipped and unclipped. **This is PPO-Clip**.

$$\text{PPO} = \min(\text{unclipped}, \text{clipped})$$

Proximal Policy Optimization (PPO)

```
x = “you prefer cats or dogs?”
for epoch in epochs:
    y = llm.generate(x)
    r = 10 if y == “cats” else 2

    with torch.no_grad():
        old_logprobs = llm(y).logprobs()

    for step in off_policy_steps:
        new_logprobs = llm(y).logprobs()
        # log (a / b) = log a - log b:
        unclipped = new_logprobs - old_logprobs
        clipped = clip(unclipped, 1-є, 1+є)
        minimum = min(clipped, unclipped)
        objective = (r * minimum).mean()
        loss = -objective
        loss.backward()
        llm.update()
```

Advantages & Kullback-Leibler Divergence

Advantages

The reward has high-variance: it is computed on a single sample.

In practice we use a reduced variance reward, we call *advantages,* noted *A.*

The advantages subtract to the reward *R(x, y)* a baseline *b: A = R(x, y) - b*

The baseline *b* is computed differently depending on the method:

- **PPO:** A = *R(x, y) - v(s)* uses a trainable neural network
- **GRPO:** Uses PPO clip loss but with a different advantages computation. Samples *G* answers per prompt, get *G* rewards: $\mathbf{r} = \{r_1, r_2, \cdots, r_G\}$ and uses the mean a baseline:

$$b = \text{mean}(\mathbf{r})$$
$$A_i = r_i - b$$

The original GRPO normalizes the advantages: $A_i = (r_i - b) / \text{std}(\mathbf{r})$.
But some works argue that this is not necessary.

Kullback-Leibler Divergence

→ In LLMs, It is common to use a penalty term that prevents the policy to *diverge* too much from the reference model (the LLM before RL Training)

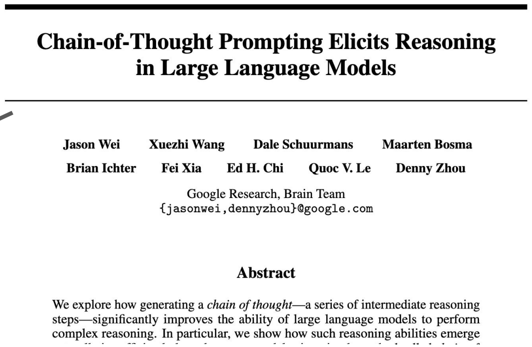
→ In general Kullback-Divergence is used. There are many ways to compute it.

→ Some works show that this is not necessary as it prevents the policy to explore more.

Reasoning LLMs with Reinforcement Learning

CoT reasoning

First author of this paper

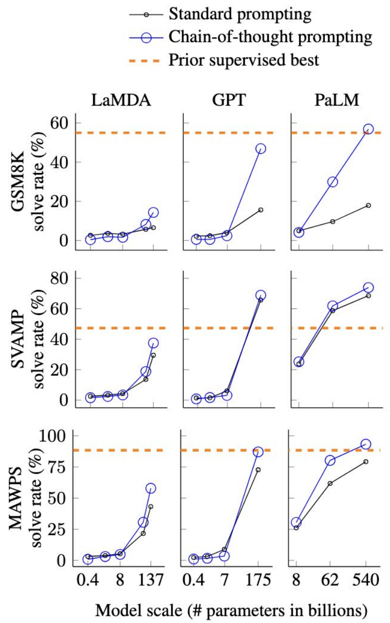


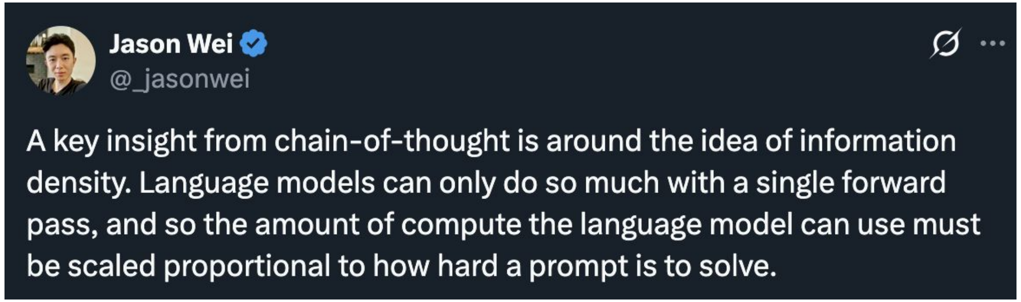
Chain-of-Thought Prompting Elicits Reasoning in Large Language Models

Jason Wei Xuezhi Wang Dale Schuurmans Maarten Bosma
Brian Ichter Fei Xia Ed H. Chi Quoc V. Le Denny Zhou
Google Research, Brain Team
{jasonwei,dennyzhou}@google.com

Abstract

We explore how generating a *chain of thought*—a series of intermediate reasoning steps—significantly improves the ability of large language models to perform complex reasoning. In particular, we show how such reasoning abilities emerge





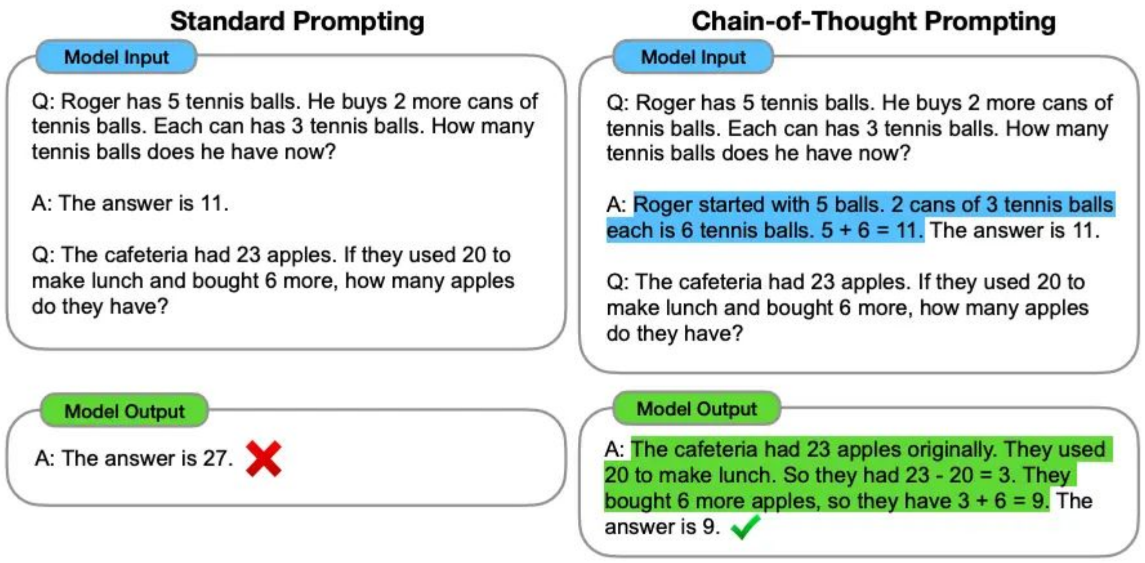
Jason Wei @_jasonwei

A key insight from chain-of-thought is around the idea of information density. Language models can only do so much with a single forward pass, and so the amount of compute the language model can use must be scaled proportional to how hard a prompt is to solve.

Some problems require more time.
The model needs more "*thinking*" tokens. The model needs more forward passes.

Chain-of-Thought: the model uses more thinking/reasoning tokens before

CoT reasoning



Standard Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27. ✗

Chain-of-Thought Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

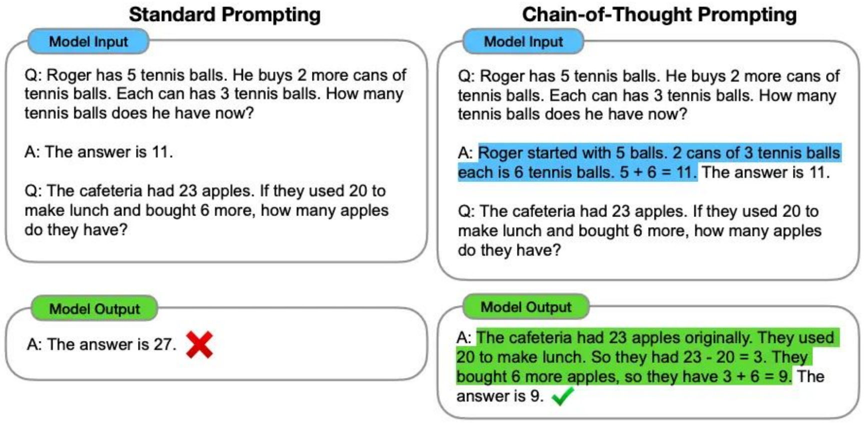
Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9. The answer is 9. ✓

This is few-shot CoT prompting, where the model is given a demonstration of reasoning before answering.

CoT reasoning



Standard Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27. ✗

Chain-of-Thought Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9. The answer is 9. ✓

This is few-shot CoT prompting, where the model is given a demonstration of reasoning before answering.

It requires manually designing the CoT strategy and/or length.

What is the best CoT prompt? There many different prompts and it is difficult to say which one is the best.

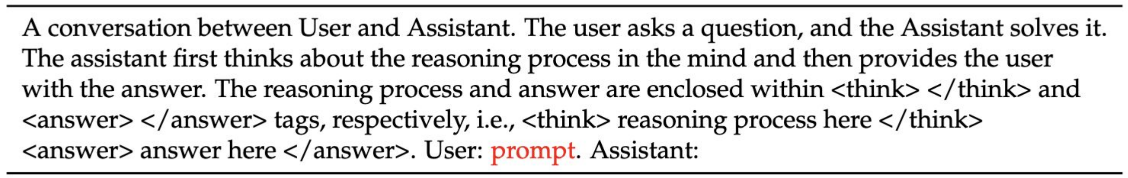
Instead of manually defining reasoning strategy with the CoT prompt, what if the model learns to generate reasoning CoT by its own?

Incentivizing LLM Reasoning with Reinforcement Learning

DeepSeek-r1-Zero

DeepSeek-r1-Zero is the first open LLM trained to reason with reinforcement learning. The base model being DeepSeek-V3-Base (a base model is the model after pretraining)

Using GRPO for reinforcement learning, the model is incentivized to reason before answering. No manual design on the CoT format or length!

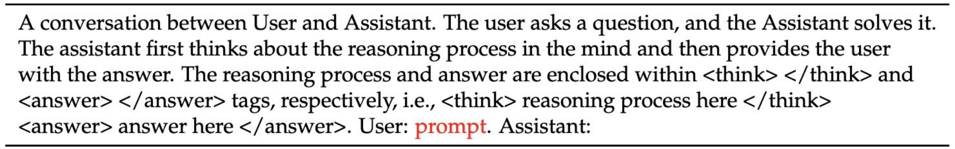


A conversation between User and Assistant. The user asks a question, and the Assistant solves it. The assistant first thinks about the reasoning process in the mind and then provides the user with the answer. The reasoning process and answer are enclosed within <think> </think> and <answer> </answer>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

Table 1 | Template for DeepSeek-R1-Zero. prompt will be replaced with the specific reasoning question during training.

Incentivizing LLM Reasoning with Reinforcement Learning

DeepSeek-r1-Zero



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:

Table 1 | Template for DeepSeek-R1-Zero. prompt will be replaced with the specific reasoning question during training.

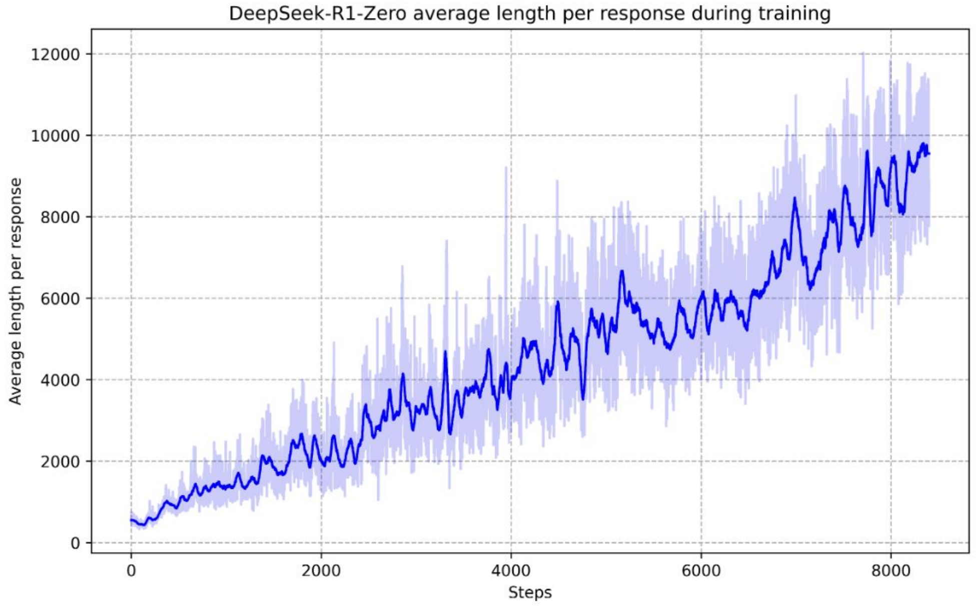
→ The model is *encouraged* to reason before answering during the RL training on verifiable tasks. That's it.

→ During RL training, the model receives two rewards:
(1) ***Outcome*** reward based on the accuracy of the answer against a verifier
(2) A format reward: whether the imposed format is respected or not

Note: an outcome reward, as opposed to ***process*** reward, only evaluates the accuracy of the answer, not the intermediate reasoning steps.

Incentivizing LLM Reasoning with Reinforcement Learning

DeepSeek-r1-Zero



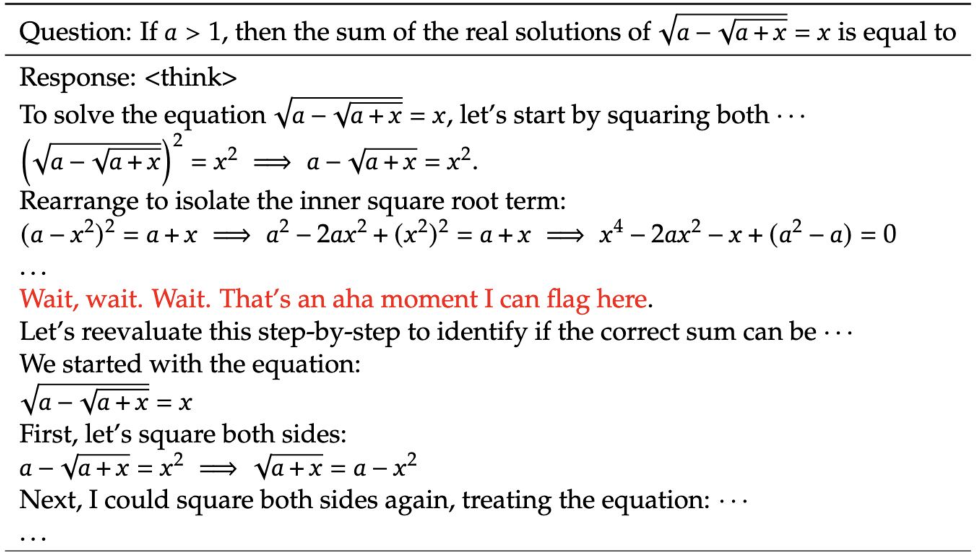
During the RL training, some *behaviors* emerge without supervision:

1. **Emergence of Long CoT**

The model understands that in order to increase the accuracy, and hence the reward, it should allocate more thinking tokens to some problems.

Incentivizing LLM Reasoning with Reinforcement Learning

DeepSeek-r1-Zero



Question: If $a > 1$, then the sum of the real solutions of $\sqrt{a - \sqrt{a+x}} = x$ is equal to

Response: <think>

To solve the equation $\sqrt{a - \sqrt{a+x}} = x$, let's start by squaring both $\cdots$

$\left(\sqrt{a - \sqrt{a+x}}\right)^2 = x^2 \implies a - \sqrt{a+x} = x^2$.

Rearrange to isolate the inner square root term:

$(a - x^2)^2 = a + x \implies a^2 - 2ax^2 + (x^2)^2 = a + x \implies x^4 - 2ax^2 - x + (a^2 - a) = 0$

$\ldots$

Wait, wait. Wait. That's an aha moment I can flag here.

Let's reevaluate this step-by-step to identify if the correct sum can be $\cdots$

We started with the equation:

$\sqrt{a - \sqrt{a+x}} = x$

First, let's square both sides:

$a - \sqrt{a+x} = x^2 \implies \sqrt{a+x} = a - x^2$

Next, I could square both sides again, treating the equation: $\cdots$

$\ldots$

During the RL training, some *behaviors* emerge without supervision:

2. **Emergence of problem-solving strategies.**

The model learns to revise its reasoning and to change its strategy when needed.

Incentivizing LLM Reasoning with Reinforcement Learning

What about generalization to other tasks?

Many large-scale reasoning LLMs are trained on some verifiable tasks: code, math, instruction following, etc.

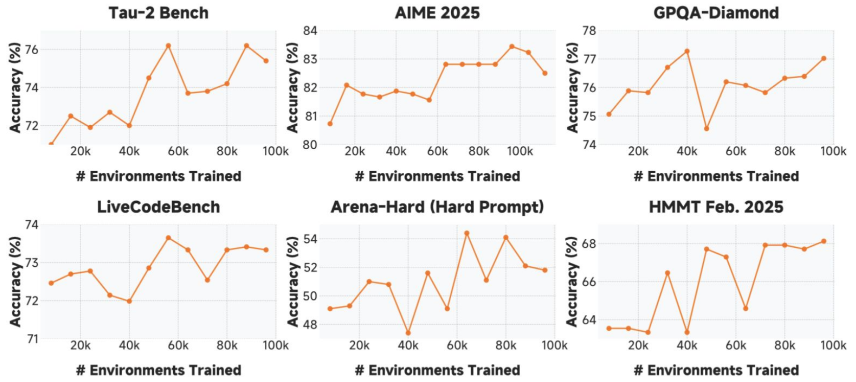


Figure 5 Generalization of code-agentic RL training to other task domains.

Mimo-v2-Flash: https://github.com/XiaomiMiMo/MiMo-V2-Flash/blob/main/paper.pdf

A reasoning agent trained on code via RL improves performance on other tasks

Incentivizing LLM Reasoning with Reinforcement Learning

What about creative tasks?

Training LLMs to be creative with Reinforcement Learning is an
active open research question

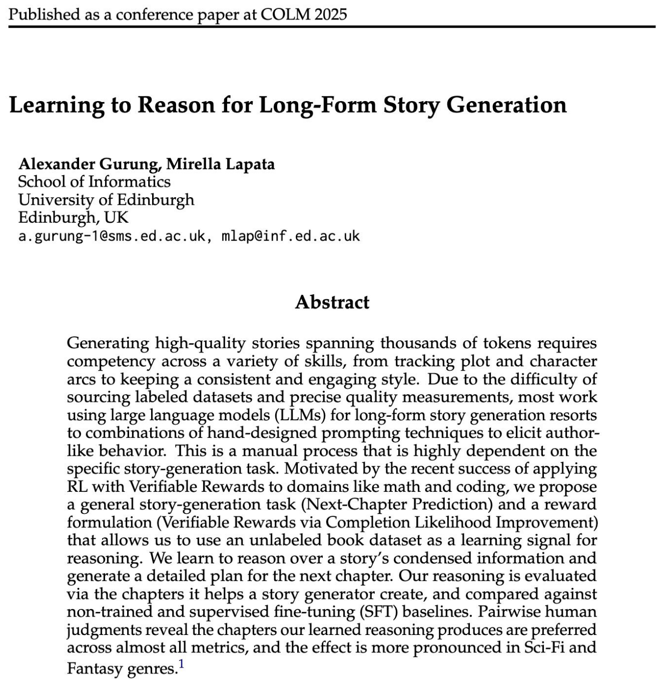
First, what is creativity? If we accept that creativity is not objectively measurable then it is difficult to design a training signal for this.

There are different approaches:

1. Train the model via RL to write *as* real authors
2. Use imperfect criterions as reward during RL training

Incentivizing LLM Reasoning with Reinforcement Learning

What about creative tasks?



Published as a conference paper at COLM 2025

Learning to Reason for Long-Form Story Generation

Alexander Gurung, Mirella Lapata
School of Informatics
University of Edinburgh
Edinburgh, UK
a.gurung-1@sms.ed.ac.uk, mlap@inf.ed.ac.uk

Abstract

Generating high-quality stories spanning thousands of tokens requires competency across a variety of skills, from tracking plot and character arcs to keeping a consistent and engaging style. Due to the difficulty of sourcing labeled datasets and precise quality measurements, most work using large language models (LLMs) for long-form story generation resorts to combinations of hand-designed prompting techniques to elicit author-like behavior. This is a manual process that is highly dependent on the specific story-generation task. Motivated by the recent success of applying RL with Verifiable Rewards to domains like math and coding, we propose a general story-generation task (Next-Chapter Prediction) and a reward formulation (Verifiable Rewards via Completion Likelihood Improvement) that allows us to use an unlabeled book dataset as a learning signal for reasoning. We learn to reason over a story's condensed information and generate a detailed plan for the next chapter. Our reasoning is evaluated via the chapters it helps a story generator create, and compared against non-trained and supervised fine-tuning (SFT) baselines. Pairwise human judgments reveal the chapters our learned reasoning produces are preferred across almost all metrics, and the effect is more pronounced in Sci-Fi and Fantasy genres.[1]

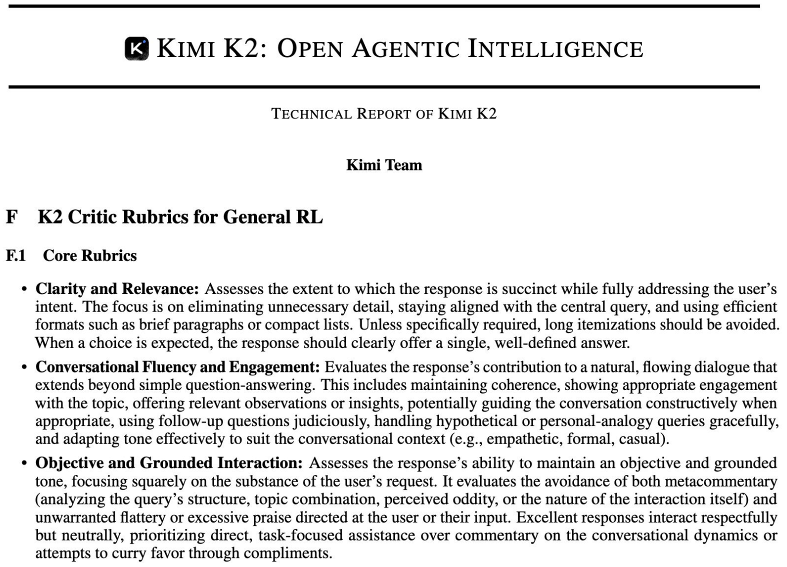
- A book is written in multiple steps.
- And at each step, the model is given a summary of the story, and is trained via RL to reason, plan and predict the next chapter.
- Reward: likelihood of the gold next-chapter given the model's reasoning compared to the likelihood without the reasoning

$$[1 - \frac{PPL_{\pi^{\mathcal{G}}}(y|x,a)}{PPL_{\pi^{\mathcal{G}}}(y|x)}] \times 100$$

Note: Perplexity (PPL) is the exponentiated negative log-likelihood

Incentivizing LLM Reasoning with Reinforcement Learning

What about creative tasks?



KIMI K2: OPEN AGENTIC INTELLIGENCE

TECHNICAL REPORT OF KIMI K2

Kimi Team

F K2 Critic Rubrics for General RL

F.1 Core Rubrics

- **Clarity and Relevance:** Assesses the extent to which the response is succinct while fully addressing the user's intent. The focus is on eliminating unnecessary detail, staying aligned with the central query, and using efficient formats such as brief paragraphs or compact lists. Unless specifically required, long itemizations should be avoided. When a choice is expected, the response should clearly offer a single, well-defined answer.
- **Conversational Fluency and Engagement:** Evaluates the response's contribution to a natural, flowing dialogue that extends beyond simple question-answering. This includes maintaining coherence, showing appropriate engagement with the topic, offering relevant observations or insights, potentially guiding the conversation constructively when appropriate, using follow-up questions judiciously, handling hypothetical or personal-analogy queries gracefully, and adapting tone effectively to suit the conversational context (e.g., empathetic, formal, casual).
- **Objective and Grounded Interaction:** Assesses the response's ability to maintain an objective and grounded tone, focusing squarely on the substance of the user's request. It evaluates the avoidance of both metacommentary (analyzing the query's structure, topic combination, perceived oddity, or the nature of the interaction itself) and unwarranted flattery or excessive praise directed at the user or their input. Excellent responses interact respectfully but neutrally, prioritizing direct, task-focused assistance over commentary on the conversational dynamics or attempts to curry favor through compliments.

Idea: We cannot verify creativity but we have some imperfect criterions for that, and a model can be evaluated against those criterions

The policy generates texts and evaluates itself against categories (***rubrics***) such as clarity, engagement, fluency, etc.

Incentivizing LLM Reasoning with Reinforcement Learning

Scaling Reinforcement Learning training

DeepSeek team don’t really show how much compute they invested in the Reinforcement Learning training, compared to the pre-training.

Usually, LLMs are trained in three steps: pretraining, mid-training, post-training.

Pre-training is where labs invest most of the compute, post-training being the cheapest part.

RL Reasoning is a new training scaling axis, that is very different from the next token prediction training. Here is why:

1. RL is said to generalize better than next token prediction training
2. RL is said to more *token-efficient (or sample efficient):* the model can learn lot with few tokens. The model can improve even after ***many many*** epochs

Incentivizing LLM Reasoning with Reinforcement Learning

Scaling Reinforcement Learning training

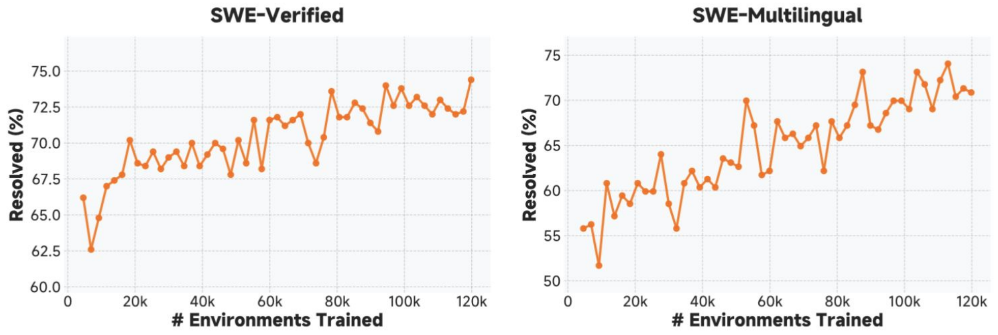


Figure 4 Code-agentic RL scaling curves. The X-axis represents total interactive environments consumed during on-policy RL rollouts; the Y-axis shows resolved rates on SWE-Bench-Verified and SWE-Bench-Multilingual.

Mimo-v2-Flash: https://github.com/XiaomiMiMo/MiMo-V2-Flash/blob/main/paper.pdf

The model is far from saturating when scaling the training time (measure as the number of interactions with the RL environments)

Close labs are probably trying to scale RL at the level of the compute invested in pre-training

Open-source folks still invest less compute, and don't scale the compute of RL training

Incentivizing LLM Reasoning with Reinforcement Learning

Long CoT Reasoning requires Long Context LLM

The goal is to have LLMs that can reason and do tasks for days, weeks, months…

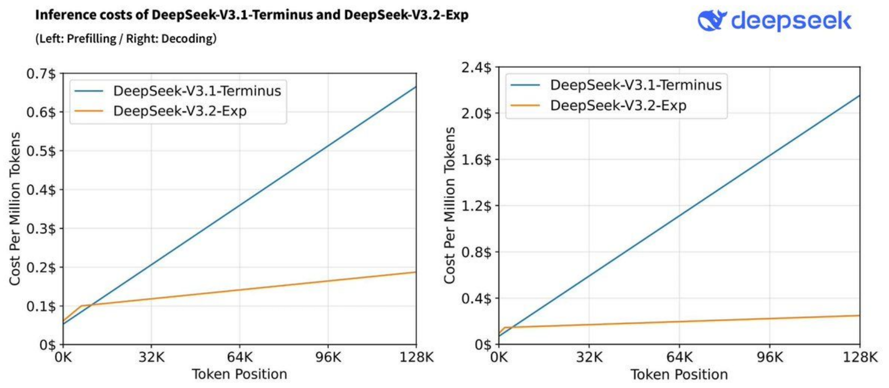
This is impossible if we use naive attention implementation, because the KV-cache memory grows linearly

Some recent solutions:

1. Hybrid Local Attention + Linear Attention

2. Compression (KV-Cache compression, Token pruning)

3. Sparse Attention (NSA, MOBA, DSA): compute attention only on selected tokens



DeepSeek Sparse Attention is used in v3.2 Exp

Incentivizing LLM Reasoning with Reinforcement Learning

Improving RL Training

- ***DAPO -*** Dynamic Sampling (https://arxiv.org/pdf/2503.14476)

  In GRPO, if all the examples in the rollouts have a reward of 1 or 0, they have a null advantage (1 - mean([1, 1, …, 1]) = 0.

  Solution: dynamically continue to generate rollouts until the batch until the advantages are not zeros.

- **Precision mismatch** (https://qingkeai.online/upload/pdf/20250830.pdf)
  In off-policy training, the rollouts are generated on a fast inference server. The probability distribution of the trainer and sampler are not the same for the same texts.

  Solution: use exactly the same operations both for the trainer and inference server

Incentivizing LLM Reasoning with Reinforcement Learning

Accelerating RL Training

The rollouts are time consuming in RL Training. It’s possible to accelerate RL training if we accelerate the rollouts generation. Here are some solutions:

- Multi-token Prediction and Speculative decoding
- Do rollouts using a quantized model but update the policy with higher precision
- Decouple rollouts generation and training in a asynchronous way.