Inference Scaling

Morgan Sinclaire

The "skin-of-an-onion" analogy is also helpful. In considering the functions of the mind or the brain we find certain operations which we can explain in purely mechanical terms. This we say does not correspond to the real mind: it is a sort of skin which we must strip off if we are to find the real mind. But then in what remains we find a further skin to be stripped off, and so on. Proceeding in this way do we ever come to the "real" mind, or do we eventually come to the skin which has nothing in it?

– A.M. Turing

Overview: Layers of an Onion

Goal: Leveraging longer inference from base LLMs

Training Setup: Verifiable rewards on CoTs

Methods Applied: RL

Inner Algorithm: PPO

Overview: Layers of an Onion

Better to understand the outer layers first!

Goal: Leveraging longer inference from base LLMs

Outline

Myopia vs. Planning

Base LLMs are good at a lot of things. One clear weakness: **multi-step reasoning / long-horizon planning**.

Running example: solving a difficult math problem.

- Have a long-term goal.
- Need to identify the right subproblems (e.g. lemmas) to reach first.
- Initial plan will often be wrong: need to backtrack and reformulate many times.

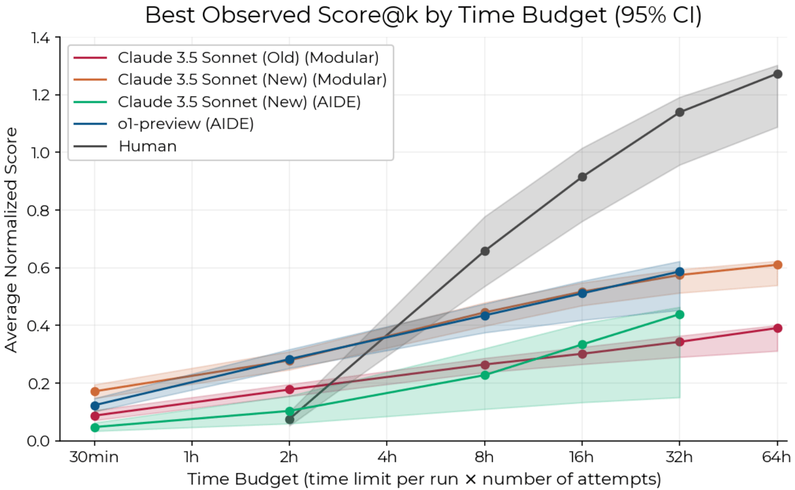
Also, what's the most likely continuation of this prompt?

`[...] and we know` $(a+b)^2 = a^2 + b^2$ `so`

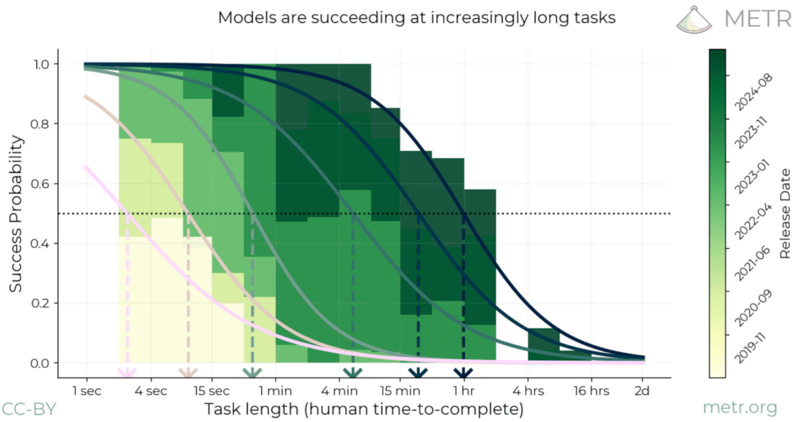
$\therefore$ Base LLMs can't backtrack when they make their first mistake (CoT prompting can help, but only so much).

Myopia vs. Planning

Given a hard problem, a base LLM will flail around a few minutes and then flatline, whereas humans have robust gains to longer thinking times (source: METR).



Long-Horizon Planning

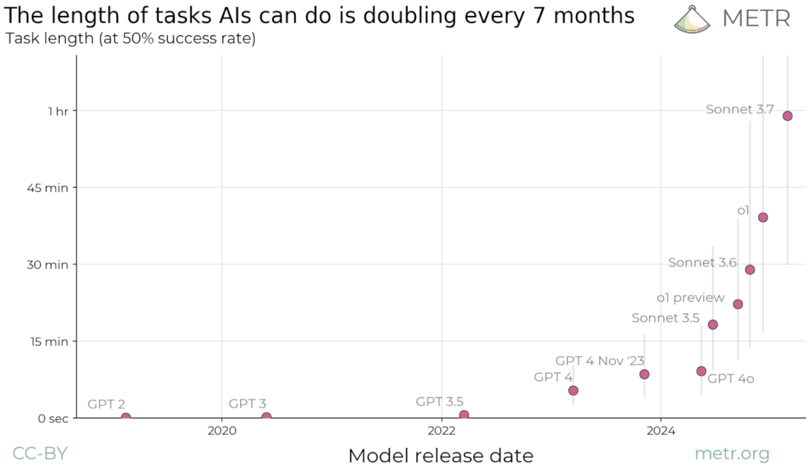


On METR's suite of SW tasks, Claude 3.7 (right-most curve) can:

- Always do ≤ 4 min tasks.
- Rarely do ≥ 2 hour tasks.
- About half the time it succeeds at 1 hour tasks.

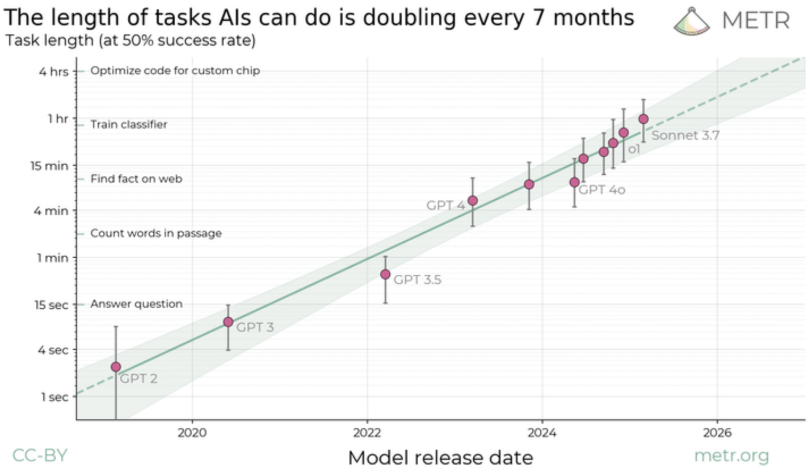
Other LLMs have similar logistic curves.

Long-Horizon Planning



This task length (@ 50% success rate) has been increasing with an obvious exponential.

Long-Horizon Planning



When plotted on a $\log$ scale, a straight line seems to fit well. (METR notes the newer models seem to have a steeper curve, but it's too early to do a proper regression.)

Myopia vs. Planning

Base LLMs have been improving at this, but slowly.[1]

Fundamentally, supervised learning (SL) on the next token is a myopic regime, and not the right kind of training for this.

On the other hand, reinforcement learning (RL) is the framework for training systems ("agents") to do long-term planning.

[1] In the limit, next-token prediction means predicting the outputs of long-term planners, but this says nothing about the empirical convergence rates.

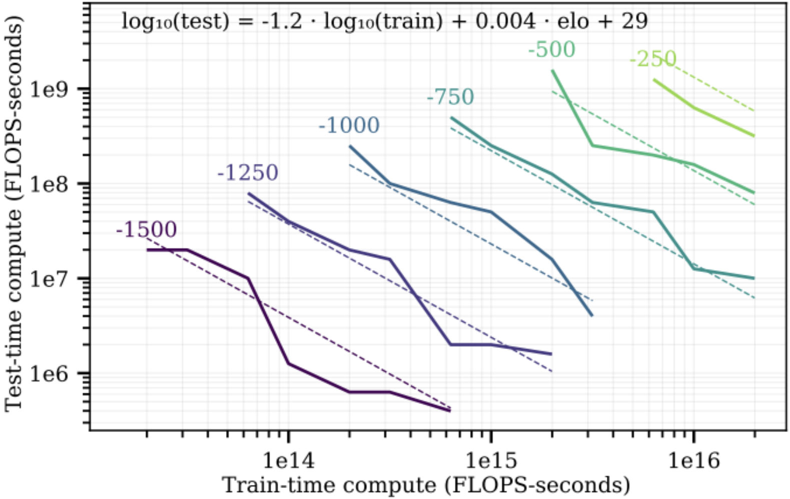
Inference Scaling

2018-2023 was the era of *pretraining scaling*: running larger models on larger next-token datasets. This is over (evidently, the data ran out).

The new paradigm is *inference scaling*: getting models to think deeper chains of thought and do longer projects.

Pretraining had specific scaling laws (FLOPs vs. perplexity) and phase transitions (e.g. in-context learning). Inference scaling will probably (?) be similar.

[Jones 2021](#) studied this in a toy setting ($n \times n$ Hex):



What is a Reasoning Model?

There's a few levels to address this:

1. At a high level, how are these different from base LLMs?
   - They're trained to carry out longer chains of inference, so they're better at multi-step reasoning/planning.
2. How is this training done?
   - With RL techniques applied to chains of thought (CoTs).
3. But how exactly?
   - Policy gradient algorithms like PPO/GRPO. These are complex but well-known.

Outline

Real RL = RLHF − HF

The type of RL used here is more in line with traditional RL ideas. RLHF is a separate thing:

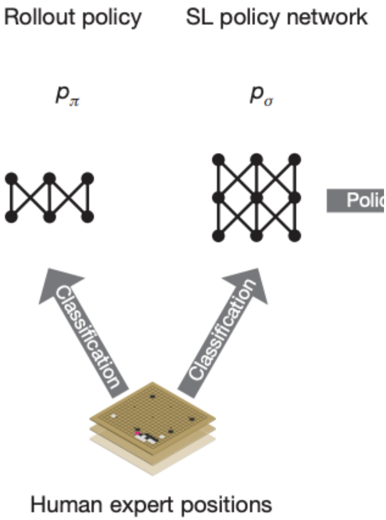
Andrej Karpathy @karpathy 7 Aug 2024

RLHF is just barely RL

Andrej Karpathy @karpathy 16 Sep 2024

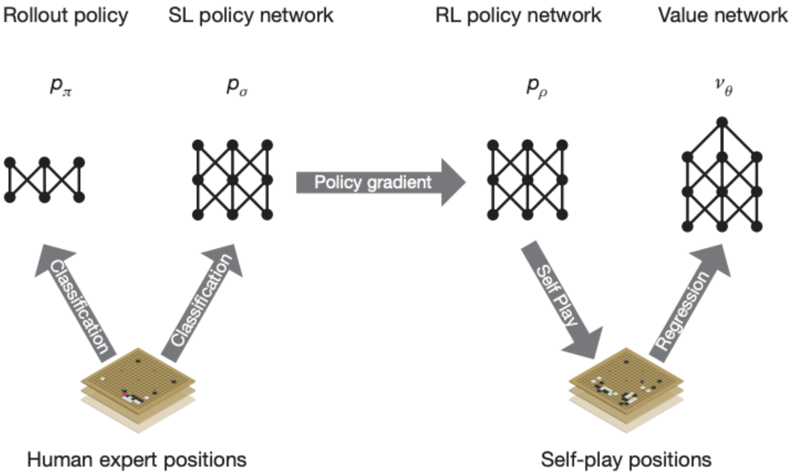
You can tell the RL is done properly when the models cease to speak English in their chain of thought

AlphaGo: SL → RL



Phase 1: SL for next-move prediction on data from human experts (to reach ∼2 dan level).

AlphaGo: SL $\rightarrow$ RL



Phase 2: RL on synthetic data from self-play (to exceed human performance).

SL vs. RL

Most LLM pretraining/fine-tuning consists of:

- Modeling-humans-modeling-reality

Rather than:

- Modeling reality itself

RL $\approx$ the class of techniques where an agent learns to maximize reward from its environment in an open-ended setting.

Input: A base LLM that can generate sometimes-coherent CoTs.

Goal: Train it to do better on problems with verifiable answers e.g. math/programming.

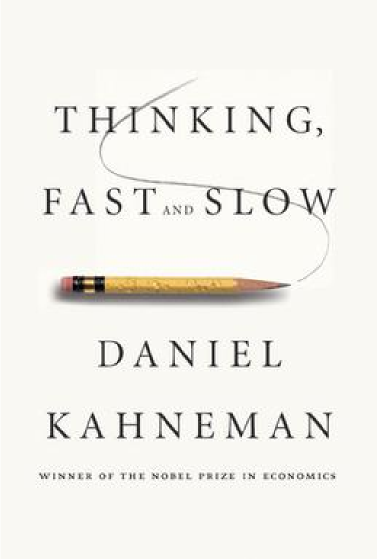
Analogy: Babbler vs. Critic

Standard Cog Psych posits:

- System 1, for intuitive/snap judgments
- System 2, for deliberate, logical thinking

As an informal analogy, maybe intelligence consists of a:

- Babbler, who blathers words-that-sound-like-me
- Critic, who vetoes illogical babblings



Outline

Overview: Layers of an Onion

Goal: Leveraging longer inference by learning System 2 reasoning

Training Setup: ?

Methods Applied: ??

Inner Algorithm: ???

Scaling Search

Richard Sutton, *The Bitter Lesson* (emphasis his):

> One thing that should be learned from the bitter lesson is the great power of general purpose methods, of methods that continue to scale with increased computation even as the available computation becomes very great. The two methods that seem to scale arbitrarily in this way are *search* and *learning*.

Scaling Search

Richard Sutton, *The Bitter Lesson* (emphasis his):

> One thing that should be learned from the bitter lesson is the great power of general purpose methods, of methods that continue to scale with increased computation even as the available computation becomes very great. The two methods that seem to scale arbitrarily in this way are *search* and *learning*.

Learning: ✓
Search: ???

Noam Brown (o1 dev) has framed reasoning LLMs as applying the bitter lesson to search.

Nathan Lambert thinks this framing is a psyop.

Scaling Search

We could just *tell* the model to:

1. Think for longer.
2. Break a problem down into multiple steps.
3. Consider multiple approaches.
4. Reflect/backtrack when an approach isn't working.

But this would go against the bitter lesson! It's better for the model to *learn* strategies like this, and not be bounded by our list.

We can also create/curate a dataset of "quality reasoning" (e.g. long CoTs, backtracking), then do supervised fine-tuning on it.

This still goes against the bitter lesson: we're limiting it to the types of reasoning humans do.

Still, a supervised phase is often needed to solve the "cold start" problem in RL:

> Previous work has heavily relied on large amounts of supervised data to enhance model performance. In this study, we demonstrate that reasoning capabilities can be significantly improved through large-scale reinforcement learning (RL), even without using supervised fine-tuning (SFT) as a cold start. Furthermore, performance can be further enhanced with the inclusion of a small amount of cold-start data. In the following sections, we present: (1) DeepSeek-R1-Zero, which applies RL directly to the base model without any SFT data, and (2) DeepSeek-R1, which applies RL starting from a checkpoint fine-tuned with thousands of long Chain-of-Thought (CoT) examples. 3) Distill the reasoning capability from DeepSeek-R1 to small dense models.

Long CoTs

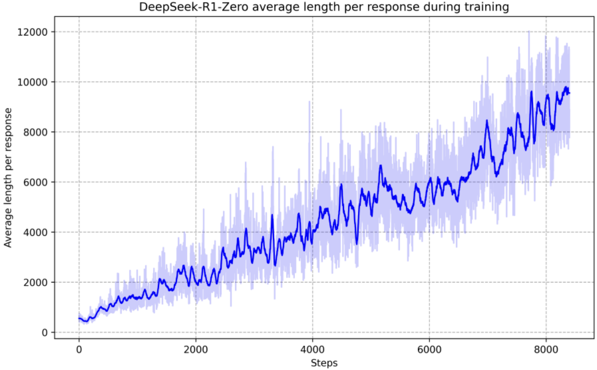


Figure 3 | The average response length of DeepSeek-R1-Zero on the training set during the RL process. DeepSeek-R1-Zero naturally learns to solve reasoning tasks with more thinking time.

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

Table 3 | An interesting "aha moment" of an intermediate version of DeepSeek-R1-Zero. The model learns to rethink using an anthropomorphic tone. This is also an aha moment for us, allowing us to witness the power and beauty of reinforcement learning.

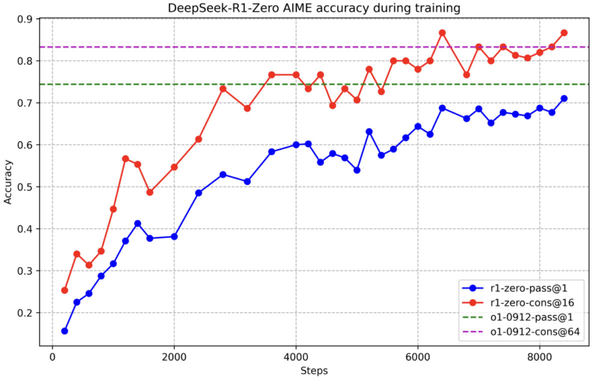


Figure 2 | AIME accuracy of DeepSeek-R1-Zero during training. For each question, we sample 16 responses and calculate the overall average accuracy to ensure a stable evaluation.

Outline

Goal: Leveraging longer inference by learning System 2 reasoning

Training Setup: Verifiable rewards on CoTs

[redacted]

Scaling Inference

How to scale up inference?

0. Shallow, non-scalable techniques.
   - Examples: CoT prompting (i.e. mental scratchpad), plurality-of-N (often used as a baseline)
1. Outcome reward model (ORM): We have a verifier that grades the outcome of a CoT, and we RL-train on this.
   - Examples: best-of-N, Monte Carlo tree search (MCTS)
2. Process reward model (PRM): We have a verifier that grades any partial CoT, and we RL-train on this.
   - Examples: beam search, diverse verifier tree search (DVTS)

Outcome Reward Models (ORM): Formatting

As a first step, we need to be able to read off the LLM's answer:

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.

> ***Format rewards**: In addition to the accuracy reward model, we employ a format reward model that enforces the model to put its thinking process between '<think>' and '</think>' tags.*

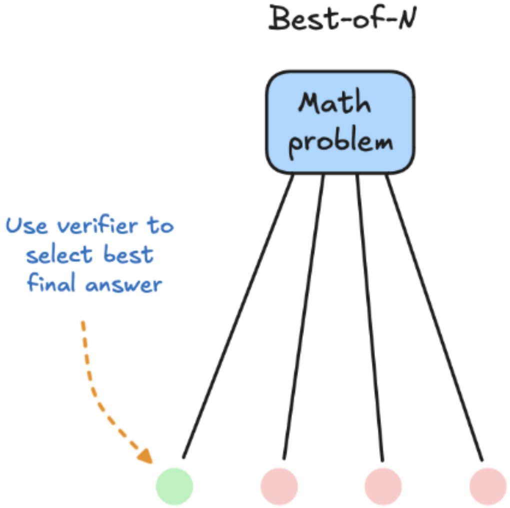
Outcome Reward Models (ORM)

How to grade the outcome of a CoT? Depends on domain:

- Math problems:
  - Known answer (e.g. competition problems): Simply grade the answer, after putting it in some normal form (e.g. $1/\sqrt{3}$ vs. $\sqrt{3}/3$).
  - General proofs: Feed into a proof assistant (e.g. Lean). Another LLM can be used to translate informal proof into desired format, or the translation can be part of the training process.
- Programming problems:
  - Test cases: Use another LLM to generate these, count $\#$ passed.

ORM: Best-of-N

In *best-of-N* (aka *guess-and-check* or *rejection sampling*) we generate N CoTs, and train on the best one(s):



Here we grade the process of the CoT, i.e. the intermediate steps (from OpenAI 2023):

The denominator of a fraction is 7 less than 3 times the numerator. If the fraction is equivalent to $2/5$, what is the numerator of the fraction? (Answer: $\boxed{14}$)

Let's call the numerator x.

So the denominator is 3x-7.

We know that x/(3x-7) = 2/5.

So 5x = 2(3x-7).

5x = 6x - 14.

So x = 7.

Figure 1: A screenshot of the interface used to collect feedback for each step in a solution.

Process Reward Models (PRM)

Let

$$x^8 + 3x^4 - 4 = p_1(x)p_2(x)\cdots p_k(x),$$

where each non-constant polynomial $p_i(x)$ is monic with integer coefficients, and cannot be factored further over the integers. Compute $p_1(1) + p_2(1) + \cdots + p_k(1)$.

I notice that the given polynomial has even degree and only even powers of x, so I can try to make a substitution to simplify it.

Let $y = x^4$, then the polynomial becomes $y^2 + 3y - 4$, which is a quadratic equation.

I can factor this quadratic equation as $(y+4)(y-1)$, so the original polynomial is $(x^4+4)(x^4-1)$.

Now I need to factor each of these factors further, if possible, over the integers.

For the first factor, $x^4 + 4$, I recall the Sophie Germain identity, which says that $a^4 + 4b^4 = (a^2 + 2b^2 + 2ab)(a^2 + 2b^2 - 2ab)$.

Applying this identity with $a = x$ and $b = 1$, I get $x^4 + 4 = (x^2 + 2x + 2)(x^2 - 2x + 2)$.

For the second factor, $x^4 - 1$, I can use the difference of squares formula, which says that $a^2 - b^2 = (a+b)(a-b)$.

Applying this formula with $a = x^2$ and $b = 1$, I get $x^4 - 1 = (x^2+1)(x^2-1)$.

Then I can apply the difference of squares formula again with $a = x$ and $b = 1$, to get $x^2 - 1 = (x+1)(x-1)$.

So the original polynomial is $(x^4+4)(x^4-1) = (x^2+2x+2)(x^2-2x+2)(x^2+1)(x+1)(x-1)$.

These are all the irreducible factors over the integers, so $p_1(x) = x^2+2x+2$, $p_2(x) = x^2-2x+2$, $p_3(x) = x^2+1$, $p_4(x) = x+1$, and $p_5(x) = x-1$.

To find the sum of their values at $x = 1$, I just need to plug in $x = 1$ into each factor and add them up.

I get $p_1(1) = 1^2 + 2\cdot 1 + 2 = 5$, $p_2(1) = 1^2 - 2\cdot 1 + 2 = 1$, $p_3(1) = 1^2 + 1 = 2$, $p_4(1) = 1 + 1 = 2$, and $p_5(1) = 1 - 1 = 0$.

So the final answer is $5 + 1 + 2 + 2 + 0 = 10$.

Answer: 10

Let

$$x^8 + 3x^4 - 4 = p_1(x)p_2(x)\cdots p_k(x),$$

where each non-constant polynomial $p_i(x)$ is monic with integer coefficients, and cannot be factored further over the integers. Compute $p_1(1) + p_2(1) + \cdots + p_k(1)$.

This problem is about factoring a polynomial over the integers, which can be a tricky task.

One possible strategy is to look for rational roots of the polynomial, using the rational root theorem.

However, that would involve checking a lot of possible factors of -4, and there is no guarantee that the polynomial has any rational roots at all.

Another possible strategy is to use some algebraic tricks to simplify the polynomial or reduce its degree.

I notice that the polynomial has only even powers of x, so I can try to substitute $y = x^2$ and see what happens.

Then the polynomial becomes $y^4 + 3y^2 - 4$, which is a quadratic equation in y^2.

I can try to factor this quadratic equation by looking for two numbers that multiply to -4 and add to 3.

Those numbers are 4 and -1, so I can write $y^4 + 3y^2 - 4 = (y^2+4)(y^2-1)$.

Now I can substitute back $x^2 = y$ and get $x^8 + 3x^4 - 4 = (x^4+4)(x^4-1)$.

This looks promising, because I have reduced the degree of the factors from 8 to 4.

I can try to factor each of these factors further, using the difference of squares formula.

I get $x^8 + 3x^4 - 4 = (x^2+2)(x^2-2)(x^2+1)(x^2-1)$.

I can apply the difference of squares formula again to the last factor and get $x^8 + 3x^4 - 4 = (x^2+2)(x^2-2)(x^2+1)(x+1)(x-1)$.

Now I have factored the polynomial completely into monic linear and quadratic factors with integer coefficients.

These are the $p_i(x)$'s that the problem is asking for.

To find the sum of their values at $x = 1$, I just need to plug in $x = 1$ into each factor and add them up.

I get $p_1(1) + p_2(1) + \cdots + p_k(1) = (1^2+2)(1^2-2)(1^2+1)(1+1)(1-1)$.

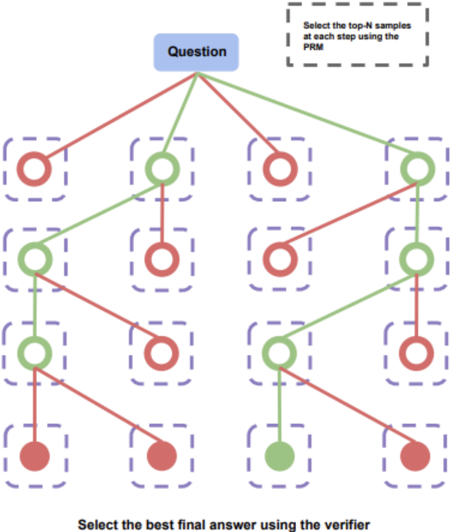
Simplifying, I get $p_1(1) + p_2(1) + \cdots + p_k(1) = (3)(-1)(2)(2)(0)$.

Multiplying, I get $p_1(1) + p_2(1) + \cdots + p_k(1) = 0$.

Answer: 0

PRM: Beam Search

If we have a PRM, we can use techniques like *beam search*. Here for some constant k, we maintain the best k paths ("beams") as we go down the search tree (e.g. $k = 2$):



Outcomes vs. Processes

How to scale up inference?

1. Outcome reward model (ORM): We have a verifier that grades the outcome of a CoT, and we RL-train on this.
    - Examples: guess-and-check, Monte Carlo tree search (MCTS)
2. Process reward model (PRM): We have a verifier that grades any partial CoT, and we RL-train on this.
    - Examples: beam search, diverse verifier tree search (DVTS)

Pros/cons of PRM vs. ORM?

Outcomes vs. Processes

Advantages of PRM over ORM:

1. Opens up more search techniques beyond best-of-N.
2. Richer reward signal, to deal with *sparse reward problem*.

Disadvantages of PRM:

1. Very hard to craft a process-based reward signal.
2. Imperfect signals often lead to drastic *reward hacking*.

Sparse Reward Problem

In most interesting cases, a random policy π_R is extremely unlikely to get any rewards at all:

- Completing a Mario/Atari level
- Playing a reasonable game of Go
- Outputting even a simple mathematical proof

Leads to a flat incentive curve where it won't learn (SL analogies: extreme class imbalance, vanishing gradients).

Common solutions:

1. *Cold start*: Somehow initialize a good policy
2. *Reward shaping*: Create intermediate rewards, i.e. PRM

Reward Hacking

Problem: In this boat racing game, my random agent ∼never reaches the finish line in the first place.

Solution: Give intermediate rewards for collecting power-ups etc.

Result

Reward Hacking

Problem: I want my virtual-robot to stack a red Lego on top of a blue Lego.

Solution: Reward based on whether:

$$\begin{aligned} &\text{height of red Lego's bottom face} \\ \approx\ &\text{height of blue Lego's top face} \end{aligned}$$

Result

Reward Hacking

Krakovna (DeepMind) has a list of 90 examples of reward hacking and specification gaming.

Irpan explains how this has long been a research obstacle in RL.

Reward Hacking: The Meta-Problem

Meta-Problem: Suppose we give an RL agent a reward function R to optimize, when this is an imperfect proxy for our "true" desires R^*. Then R, R^* will diverge with stronger optimization pressure, i.e. the agent will find creative solutions which are worse and worse for R^*.

Mechanism: The reward shapers convince themselves $R \approx R^*$ from considering a few simple behaviors. Therefore R is really just a hard-coding of "R^* for simple, uncreative agents."

Risk factors: Stronger agents, more general domains.

Outcomes vs. Processes

DeepSeek-R1 used the simplest possible ORM:

2.2.2. *Reward Modeling*

The reward is the source of the training signal, which decides the optimization direction of RL. To train DeepSeek-R1-Zero, we adopt a rule-based reward system that mainly consists of two types of rewards:

- **Accuracy rewards**: The accuracy reward model evaluates whether the response is correct. For example, in the case of math problems with deterministic results, the model is required to provide the final answer in a specified format (e.g., within a box), enabling reliable rule-based verification of correctness. Similarly, for LeetCode problems, a compiler can be used to generate feedback based on predefined test cases.
- **Format rewards**: In addition to the accuracy reward model, we employ a format reward model that enforces the model to put its thinking process between '<think>' and '</think>' tags.

We do not apply the outcome or process neural reward model in developing DeepSeek-R1-Zero, because we find that the neural reward model may suffer from reward hacking in the large-scale reinforcement learning process, and retraining the reward model needs additional training resources and it complicates the whole training pipeline.

They also report in more detail (Section 4.2) on why both PRM and MCTS didn't work.

Outcomes vs. Processes

Advantages of PRM over ORM:

1. Opens up more search techniques beyond best-of-N.
2. Richer reward signal, to deal with sparse reward problem.
   - Rumor mill: RL-on-CoT hadn't worked before *because* the base models just weren't good enough (aka we eventually solved cold start the hard way).

Disadvantages of PRM:

1. Very hard to craft a process-based reward signal.
2. **Imperfect signals often lead to drastic reward hacking.**
   - Dealbreaker

Overview: Layers of an Onion

**Goal: Leveraging longer inference
by learning System 2 reasoning**

Training Setup: Verifiable rewards on CoTs

[redacted]

Summary

Goal: We want to train our LLMs to make better use of longer inference times (e.g. carry out longer

- Some desirable "System 2" behaviors: longer CoTs, considering multiple approaches, evaluating whether it's "on the right track", backtracking

Setup: In math/programming, can (RL-)train on which CoTs worked using a simple ORM setup.
Result: Reasoning LLMs increasingly exhibit these System 2 behaviors in their (RL-)training, *without* humans telling them to.

- This includes internalizing the verifier ("inner critic"). Big question: how far will this generalize outside math/programming?