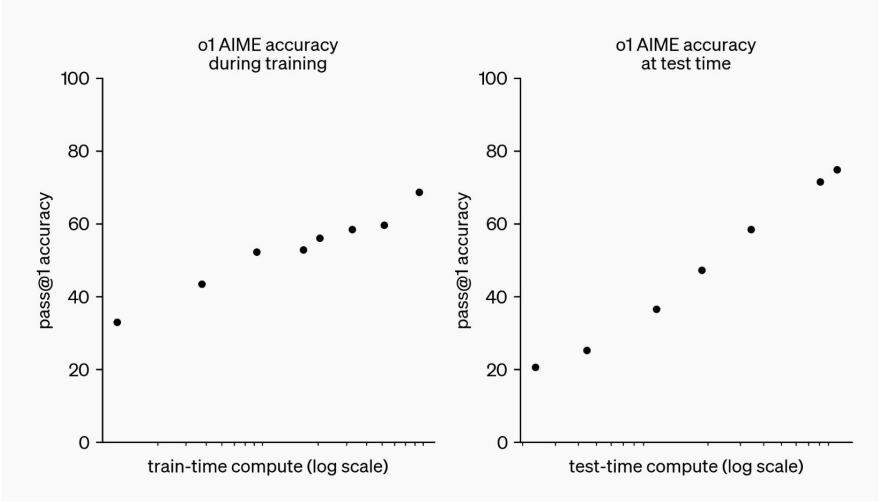
Training LLMs to Reason

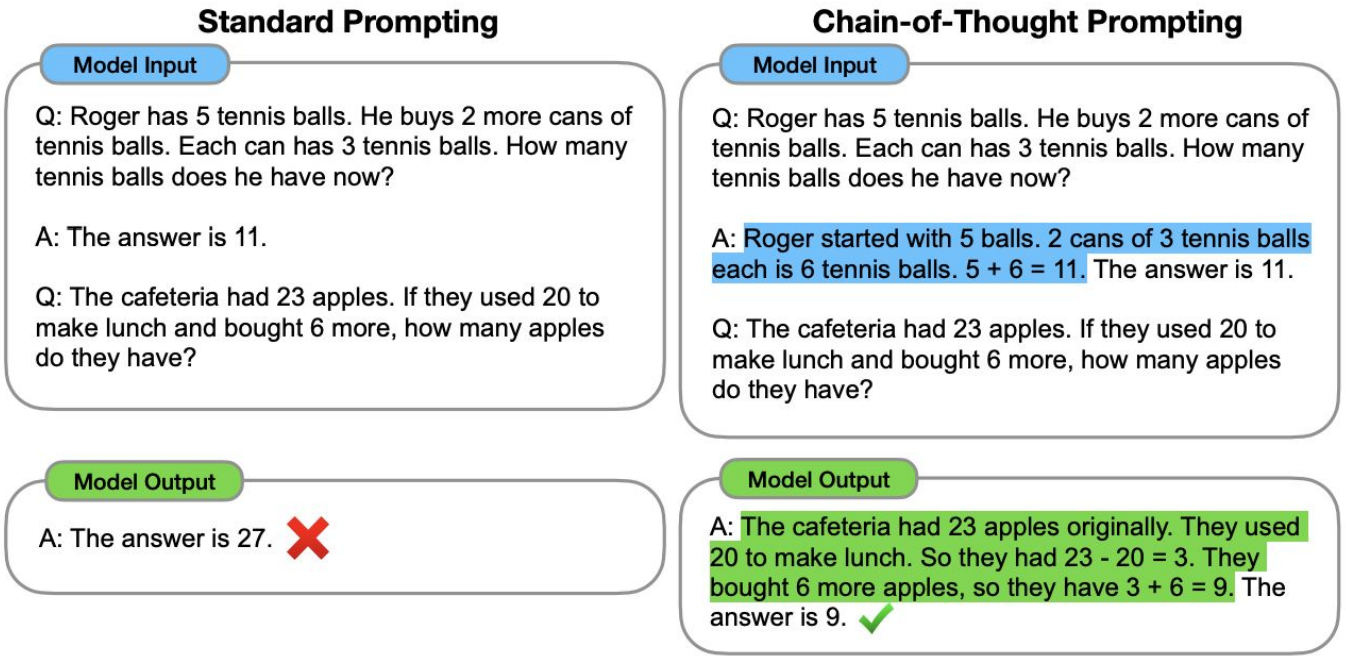
Tianbing Xu
Jan. 2025

Learning to reason with LLMs (OpenAI)

Our large-scale **reinforcement learning** algorithm teaches the model how to think productively using its **chain of thought** in a highly *data-efficient training* process.



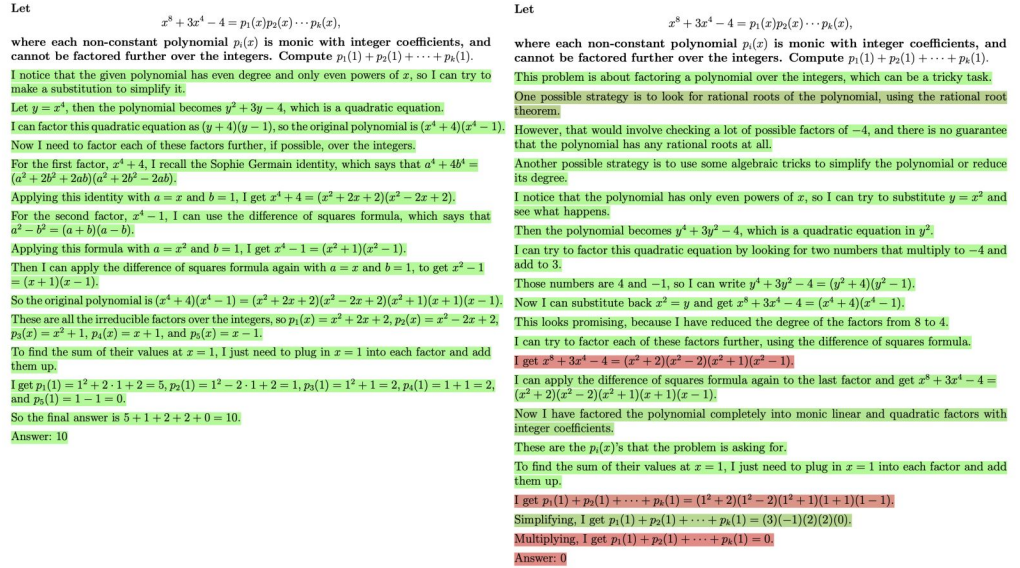
Chain-of-Thoughts (Wei et. al. NeurIPS 2022)



O1 Chain of Thoughts (OpenAI)

Through reinforcement learning, o1 learns to hone its chain of thought and refine the strategies it uses. It learns to recognize and **correct its mistakes**. It learns to **break down tricky steps into simpler ones**. It learns to **try a different approach** when the current one isn't working.

Verifier or Reward (Lightman et. al. OpenAI, ICLR 2024)



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

Figure 2: Two solutions to the same problem, graded by the PRM. The solution on the left is correct while the solution on the right is incorrect. A green background indicates a high PRM score, and a red background indicates a low score. The PRM correctly identifies the mistake in the incorrect solution.

Self-Taught Reasoner (Zelikman et. al. NeurIPS 2022)

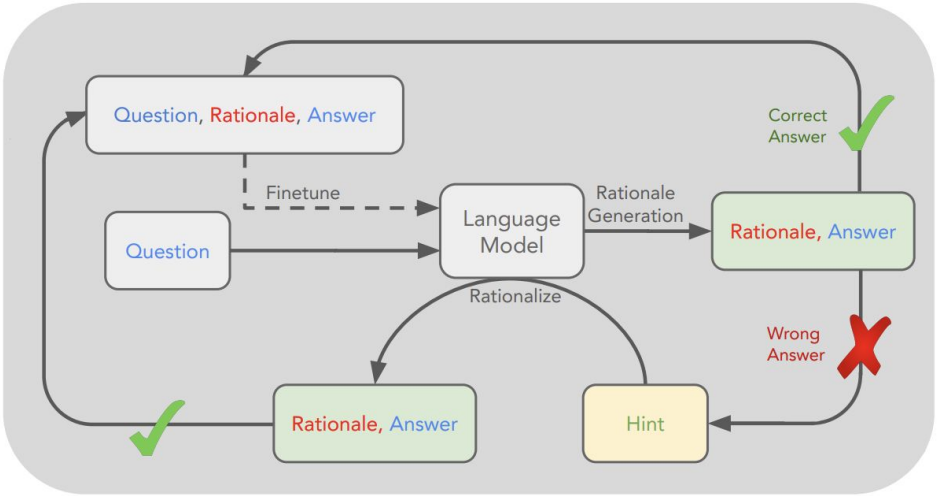


Figure 1: An overview of STaR and a STaR-generated rationale on CommonsenseQA. We indicate the fine-tuning outer loop with a dashed line. The questions and ground truth answers are expected to be present in the dataset, while the rationales are generated using STaR.

STaR Algorithm

Algorithm 1 STaR

Input M: a pretrained LLM; dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{D}$ (w/ few-shot prompts)

$M_0 \leftarrow M$ # Copy the original model
for n **in** $1...N$ **do** # Outer loop
  $(\hat{r}_i, \hat{y}_i) \leftarrow M_{n-1}(x_i) \quad \forall i \in [1, D]$ # Perform rationale generation
  $(\hat{r}_i^{\text{rat}}, \hat{y}_i^{\text{rat}}) \leftarrow M_{n-1}(\text{add\_hint}(x_i, y_i)) \quad \forall i \in [1, D]$ # Perform rationalization
  $\mathcal{D}_n \leftarrow \{(x_i, \hat{r}_i, y_i) \mid i \in [1, D] \wedge \hat{y}_i = y_i\}$ # Filter rationales using ground truth answers
  $\mathcal{D}_n^{\text{rat}} \leftarrow \{(x_i, \hat{r}_i^{\text{rat}}, y_i) \mid i \in [1, D] \wedge \hat{y}_i \neq y_i \wedge \hat{y}_i^{\text{rat}} = y_i\}$ # Filter rationalized rationales
  $M_n \leftarrow \text{train}(M, \mathcal{D}_n \cup \mathcal{D}_n^{\text{rat}})$ # Finetune the original model on correct solutions - inner loop
end for

Expert Iteration Framework (Anthony et. al. NIPS 2017)

The Imitation Learning step is analogous to a human improving their intuition for the task by studying example problems, while the expert improvement step is analogous to a human using their improved intuition to guide future analysis.

Algorithm 1 Expert Iteration

1: $\hat{\pi}_0$ = initial_policy()
2: π_0^* = build_expert($\hat{\pi}_0$)
3: **for** i = 1; i $\leq$ max_iterations; i++ **do**
4: S_i = sample_self_play($\hat{\pi}_{i-1}$)
5: $D_i = \{(s, \text{imitation\_learning\_target}(\pi^*_{i-1}(s)))|s \in S_i\}$
6: $\hat{\pi}_i$ = train_policy(D_i)
7: π_i^* = build_expert($\hat{\pi}_i$)
8: **end for**

RL Policy Gradient

STaR can be seen as an approximation to an RL-style policy gradient objective. To see this, note that M can be viewed as a discrete latent variable model $p_M(y \mid x) = \sum_r p(r \mid x)p(y \mid x, r)$; in other words, M first samples a latent rationale r before predicting y. Now, given the indicator reward function $\mathbb{1}(\hat{y} = y)$, the total expected reward across the dataset is

$$J(M, X, Y) = \sum_i \mathbb{E}_{\hat{r}_i, \hat{y}_i \sim p_M(\cdot|x_i)} \mathbb{1}(\hat{y}_i = y_i), \tag{1}$$

$$\nabla J(M, X, Y) = \sum_i \mathbb{E}_{\hat{r}_i, \hat{y}_i \sim p_M(\cdot|x_i)} \left[\mathbb{1}(\hat{y}_i = y_i) \cdot \nabla \log p_M(\hat{y}_i, \hat{r}_i \mid x_i)\right], \tag{2}$$

Reinforced Self-Training (Gulcehre et. al. Deepmind 2023)

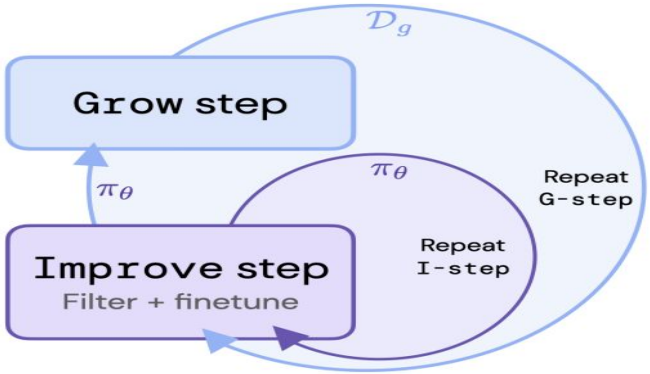


Figure 1 | ***ReST* method.** During Grow step, a policy generates a dataset. At Improve step, the filtered dataset is used to fine-tune the policy. Both steps are repeated, Improve step is repeated more frequently to amortise the dataset creation cost.

Tree-of-Thoughts (Yao et. al. NeurIPS 2024)

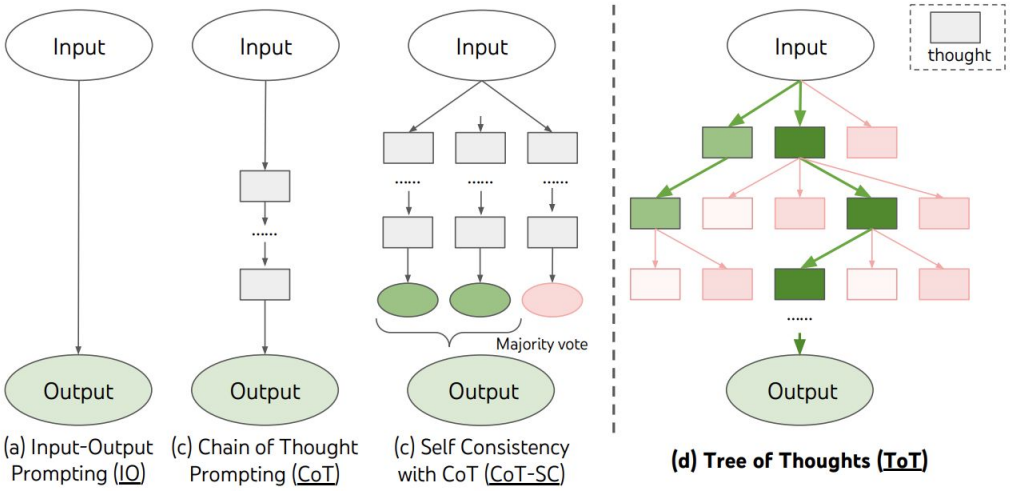


Figure 1: Schematic illustrating various approaches to problem solving with LLMs. Each rectangle box represents a *thought*, which is a coherent language sequence that serves as an intermediate step toward problem solving. See concrete examples of how thoughts are generated, evaluated, and searched in Figures 2,4,6.

Stream of Search (Gandhi et. al. COLM 2024)

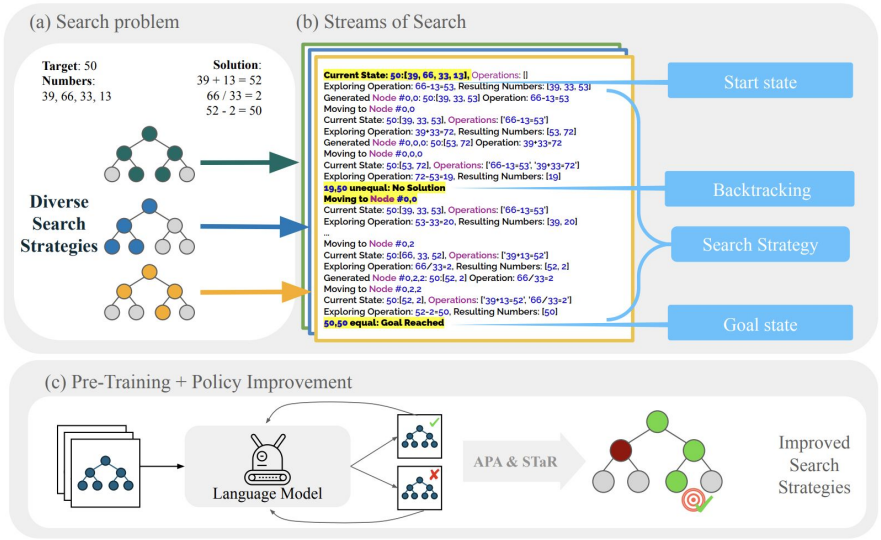


Figure 1: Overview of the Stream of Search (SoS) framework. (a) A search problem in Countdown is instantiated with input numbers and a target number. The input numbers need to be combined with simple arithmetic operations to get to the target. (b) The Stream of Search dataset contains search trajectories generated by diverse search strategies, including exploration and backtracking. (c) The language model is first trained on the SoS dataset and then iteratively improved using policy improvement techniques such as APA and STaR.

Learning to Search

(Exploration, Expansion, Prune, Backtracking, Learning from Mistakes)

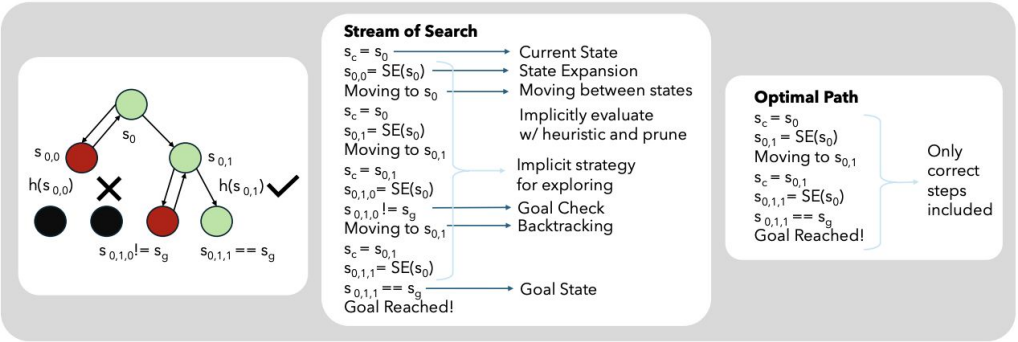


Figure 2: A visualization of how a search process is translated into a stream of search with a language for search. (left) The search process represented as a tree, with different states and operations. The colored states represent the search trajectory $\mathcal{T}$, the green states represent the correct path to the goal $\mathcal{P}$ and the arrows represent transitions between the states. The black circles represent unexplored states. (center) The search process serialized as text to create a stream of search. The labels specify the different components of the process. See Fig. 1b for how this is realized in Countdown. (right) The optimal path, $\mathcal{P}$, to the goal state. Backtracking, exploration and the messy process of search are excluded.

Test-Time Compute (Snell et. al. Deepmind 2024)

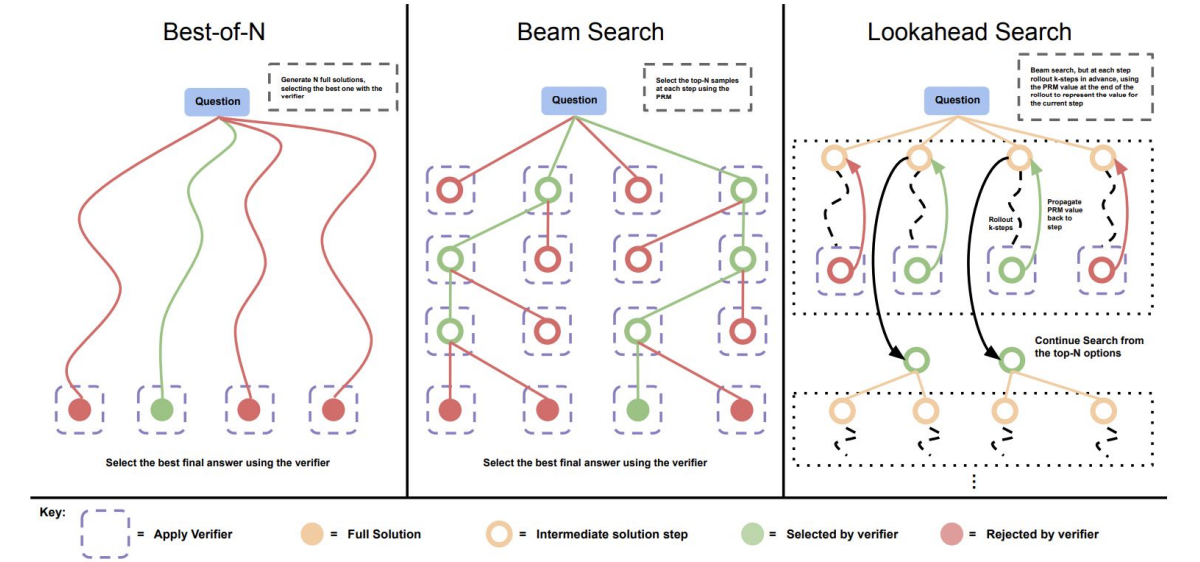


Figure 2 | ***Comparing different PRM search methods.*** **Left:** Best-of-N samples N full answers and then selects the best answer according to the PRM final score. **Center:** Beam search samples N candidates at each step, and selects the top M according to the PRM to continue the search from. **Right:** lookahead-search extends each step in beam-search to utilize a k-step lookahead while assessing which steps to retain and continue the search from. Thus lookahead-search needs more compute.

Reference

1. Learning to reason with LLMs
2. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models
3. LET'S VERIFY STEP BY STEP
4. Tree of Thoughts: Deliberate Problem Solving with Large Language Models
5. STaR: Self-Taught Reasoner Bootstrapping Reasoning With Reasoning
6. Thinking Fast and Slow with Deep Learning and Tree Search
7. Reinforced Self-Training (ReST) for Language Modeling
8. Stream of Search (SoS): Learning to Search in Language
9. Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters