CS769 Deep Learning for NLP

Reasoning and Verification in LLMs

Junjie Hu

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

https://junjiehu.github.io/cs639-spring26/

Goal for Today

- LLM Reasoning Problem, Reinforcement Learning
- Reasoning with Verifiable Rewards
    - Progress Reward Supervision
    - Automated Progress Supervision
- Reasoning without Verifiable Rewards

Part 1: The Reasoning Problem

What is Reasoning for LLMs?

- It's the ability to go beyond simple pattern matching or retrieval.
    - It involves **multi-step, logical inference** to solve novel and complex problems.
    - It's about decomposing a problem, working through intermediate steps, and arriving at a sound conclusion.
    - It's the difference between *knowing* a fact and *deriving* a new one.

Why is Reasoning Hard for LLMs?

- **"Plausible" vs. "Correct":** LLMs are trained to predict the next token, which often leads to answers that *sound plausible* but are logically flawed.

- **Logical Fallacies:** They can "hallucinate" logical steps or make simple arithmetic errors in the middle of a complex problem.

- **Error Compounding:** A single small mistake early in a reasoning chain can cascade, leading to a completely incorrect final answer.

- **The "Scaling Law" Limit:** Simply making models bigger improves knowledge, but not necessarily the *rigor* of their reasoning.

Example: A Reasoning Failure

- **The Problem**: "If a t-shirt and a hat cost $110 in total, and the t-shirt costs $100 more than the hat, how much does the hat cost?”

- **The Common (Wrong) LLM Answer**: “The hat costs $10.”

- **Why it's Wrong**:

  - If Hat = $10, then T-Shirt = $10 + $100 = $110.
    Total = $110 (T-Shirt) + $10 (Hat) = $120. (Incorrect)

- **Correct Answer**: The hat costs $5.

Key Reasoning Tasks We'll Discuss

1. Math Reasoning

Solving problems from benchmarks like MATH and GSM8K. Requires symbolic understanding and step-by-step accuracy.

2. Automatic Theorem Proving

Formal logic, proving mathematical theorems. Requires rigorous, verifiable, and logically sound steps.

3. General Reasoning

Complex, ambiguous domains like law, science, or medicine, where there may be no single verifiable answer.

Reasoning Task 1: Math Reasoning

- **Goal:** Solve complex, multi-step math problems.
- **Benchmarks**:
  - **GSM8K:** Grade School Math (thousands of problems).
  - **MATH:** High school and university competition-level math.
- **Challenge:** Requires symbolic manipulation, arithmetic, and logical deduction. A single error fails the problem.

Reasoning Task 2: Automatic Theorem Proving (ATP)

- **Goal:** Generate a formal proof for a mathematical conjecture.

- **Environment:** Formal systems like Lean, Isabelle, or Coq.

- **Challenge:** The search space is infinite. The steps must be 100% logically sound and verifiable by a proof checker.

- This is the most rigorous form of reasoning.

Reasoning Task 3: General Reasoning

- **Goal:** Provide a well-reasoned answer in an open-ended domain.
- Examples:
  - "Analyze the legal precedent for this case."
  - "Propose a differential diagnosis for this patient."
  - "Evaluate the economic impact of this policy."
- **Challenge:** Answers are complex, ambiguous, and have **no single verifiable answer**. How do we know if the reasoning is "good"?

The Core Challenge: Supervision

How do we teach a model to "think" correctly?

If we only show it the final answer, it might learn to "get lucky" instead of learning the *process*. This is the central question of reasoning alignment.

Two Types of Supervision

Outcome Supervision (OS)

Provides feedback **only on the final answer.**

"Was the final answer correct? Yes/No"

Process Supervision (PS)

Provides feedback **on each intermediate step.**

"Was that step logical? Yes/No"

Supervision 1: Outcome Supervision (OS)

- Also known as "RL from final answer" or "Outcome Reward Models (ORM)".
- The model generates a full solution (e.g., 10 steps).
- It only receives a reward (e.g., +1) if the *final* answer is correct.
- **Problem:** This is a very sparse reward. The model doesn't know *which* of the 10 steps was wrong.

Supervision 2: Process Supervision (PS)

- Also known as "RL from intermediate steps" or "Process Reward Models (PRM)".
- The model generates one step at a time.
- It receives a reward *after each step*.
- **Benefit:** This provides a dense, granular feedback signal. The model learns *exactly* where it made a mistake, right when it makes it.

Part 2: Reasoning with Verifiable Reward

Let’s Verify Step by Step

Lightman et al. (OpenAI), 2023

- "We conduct our own investigation, finding that process supervision significantly outperforms outcome supervision for training models to solve problems from the challenging MATH dataset.”

- “Lucky Guess” Problem of Outcome Supervision: A model can get the right answer for the **wrong reasons**.

  - Example: `2 * x = 10`, so `x = 10 - 2 = 8`. Wait, that's not right. `x = 10 / 2 = 5`. Oh, the answer is 5.

    - An OS model might just output "5" and get a reward.

    - It *never learns* that the `10 - 2` step was wrong.

  - This "lucky" reasoning is brittle and does **not generalize** to new problems.

The Solution: Process Supervision (PS)

- Reward the *Process*, Not the *Outcome*
    - PS teaches the model **how to think**, not just what to answer.
    - By rewarding each correct intermediate step, the model learns a robust, generalizable reasoning process.
    - It explicitly penalizes logical fallacies, even if the final answer *could* have been guessed correctly.

"Teacher": Process Reward Model (PRM)

- We can't have a human label every step in real-time. That's too slow for RL.
- **Solution:** First, we train a "teacher" model to learn the human's preferences. This teacher is the **Process Reward Model (PRM)**.
- The PRM's only job is to look at a *single* reasoning step and output a score: "Is this step correct and logical?"

PRM800K Dataset

Lightman et al. (OpenAI), 2023

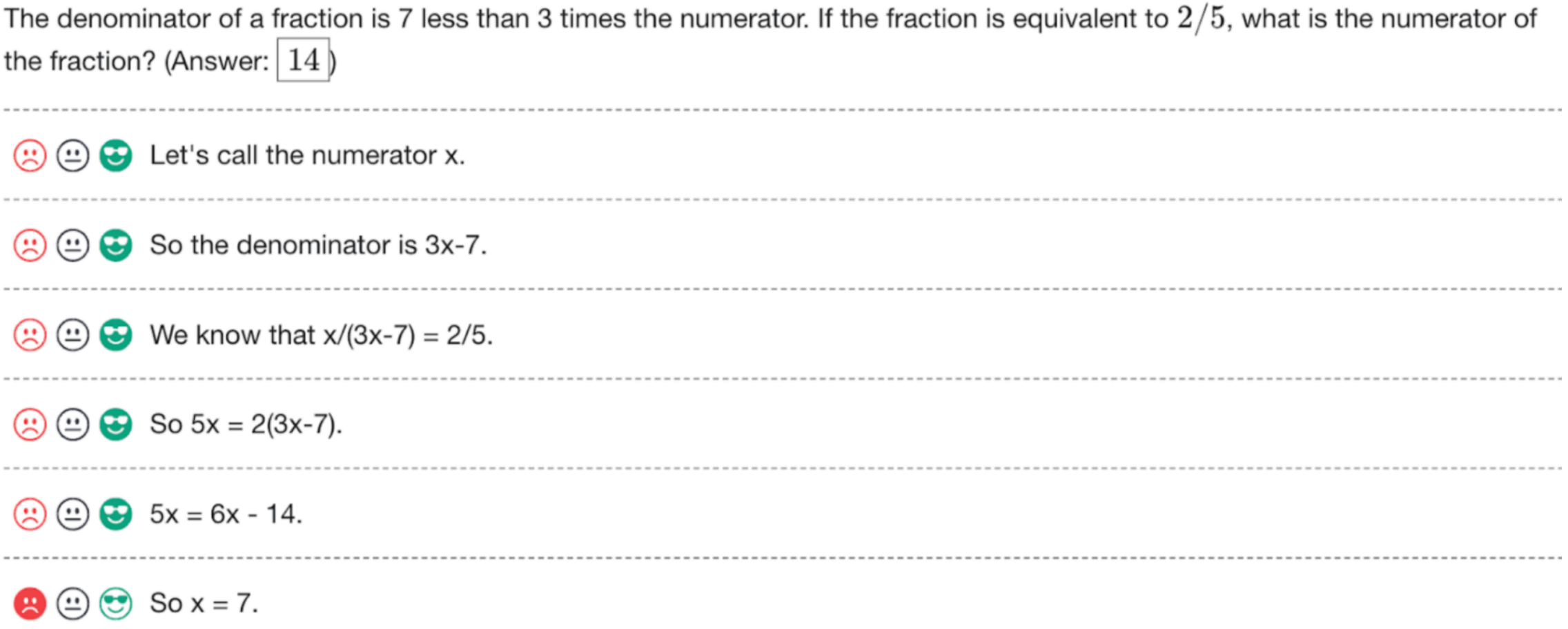
- To train the PRM, the authors created a massive dataset of human-labeled *steps*.

- **PRM800K:** 800,000 step-level feedback labels.

- Human labelers marked each step in a solution as:
  - **Positive:** Correct and moves toward the solution
  - **Neutral:** Correct, but doesn't make progress.
  - **Negative:** Incorrect or illogical.

Annotation Example

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



PRM teacher for Grading

- Two solutions to the same problem graded by the 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 $p_1(1) + p_2(1) + \cdots + p_k(1) = (1^2 + 2)(1^2 - 2)(1^2 + 1)(1 + 1)(1 - 1)$.

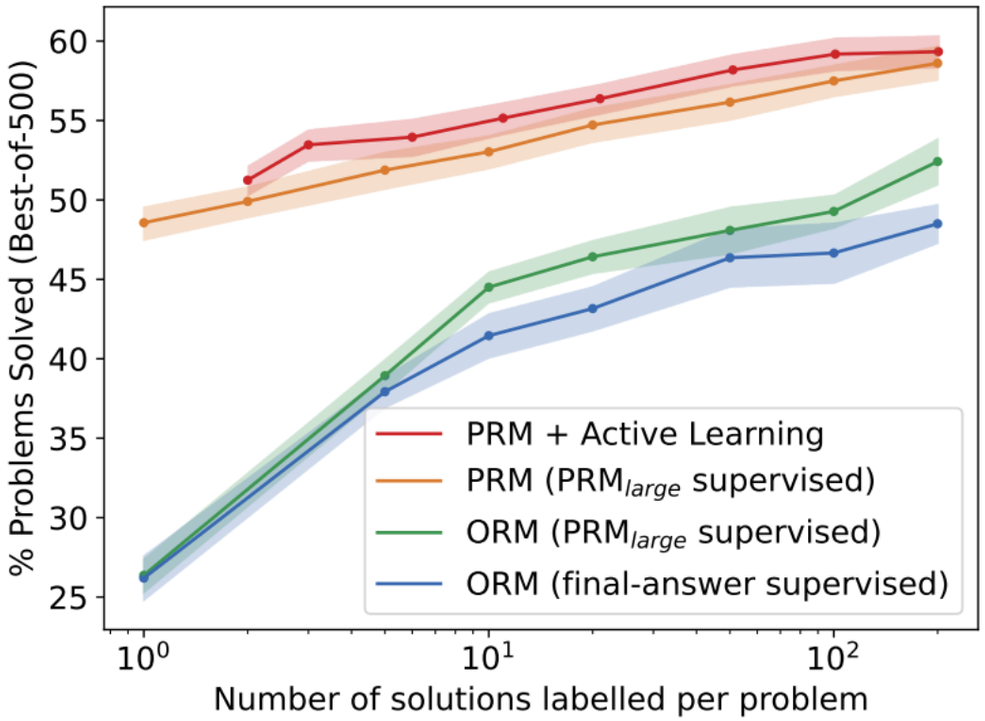
Simplifying, I get $p_1(1) + p_2(1) + \cdots + p_k(1) = (3)(-1)(2)(2)(0)$.

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

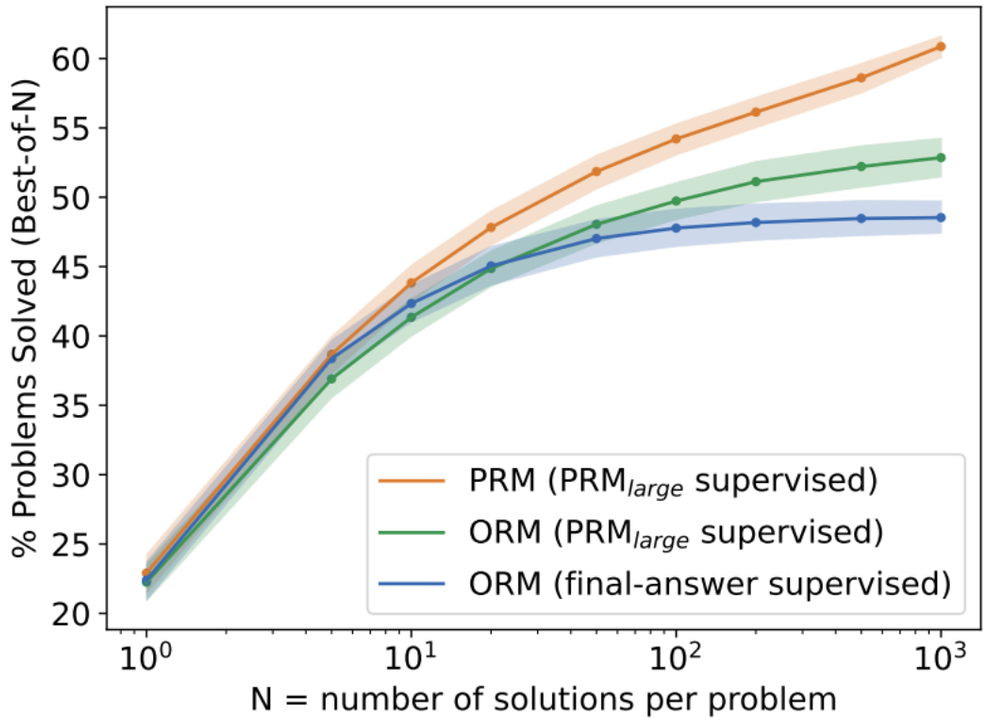
Answer: 0

PRM vs ORM

- Compare different forms of PRM and ORM: PRM works better than ORM in solving more math problems



(a) Four series of reward models trained using different data collection strategies, compared across training sets of varying sizes.



(b) Three reward models trained on 200 samples/problem using different forms of supervision, compared across many test-time compute budgets.

Limitation of PRM

- This is a fantastic result, but there's a huge problem

Creating the PRM800K dataset was **"prohibitively expensive."**

It required immense human effort to label 800,000 individual steps. How can we ever scale this to all domains?

Automated Process Supervision

Luo et al. 2024, Google DeepMind

- **The Goal:** Automate the creation of the Process Reward Model (PRM) to remove the human bottleneck.
- **Solution**: **OmegaPRM**, which is a Divide-and-conquer style Monte Carlo Tree Search (MCTS)
  - Process Annotation with Monte Carlo
  - Monte Carlo Tree Search

Process Annotation with Monte Carlo

- A model takes a question q and a prefix solution comprising the first t steps $x_{1:t}$, and outputs the completion (a.k.a "rollout" in RL) of the subsequent steps until the final answer is reached.
- For any step of a solution, the model randomly samples k rollouts from that step. The final answers of these rollouts are compared to the golden answer, providing k labels of answer correctness corresponding to the k rollouts.

$$c_t = \text{MonteCarlo}(q, x_{1:t}) = \frac{\text{num(correct rollouts from } t\text{-th step)}}{\text{num(total rollouts from } t\text{-th step)}}$$

Process Annotation with Monte Carlo

- The ratio of correct rollouts to total rollouts from the t-th step estimates the "correctness level" of the prefix steps up to t

Problem: Let p(x) be a monic polynomial of degree 4. Three of the roots of p(x) are 1, 2, 3. Find p(0)+p(4).

Golden Answer: 24

Solution: Since three of the roots of $p(x)$ .... Final Answer 20. ✗

Problem: ....
Since three of the roots of $p(x)$ are 1, 2 and 3, we can write:

Rollout 1: .... Final Answer 24. ✓

Rollout 2: .... Final Answer 24. ✓

Rollout 3: .... Final Answer 20. ✗

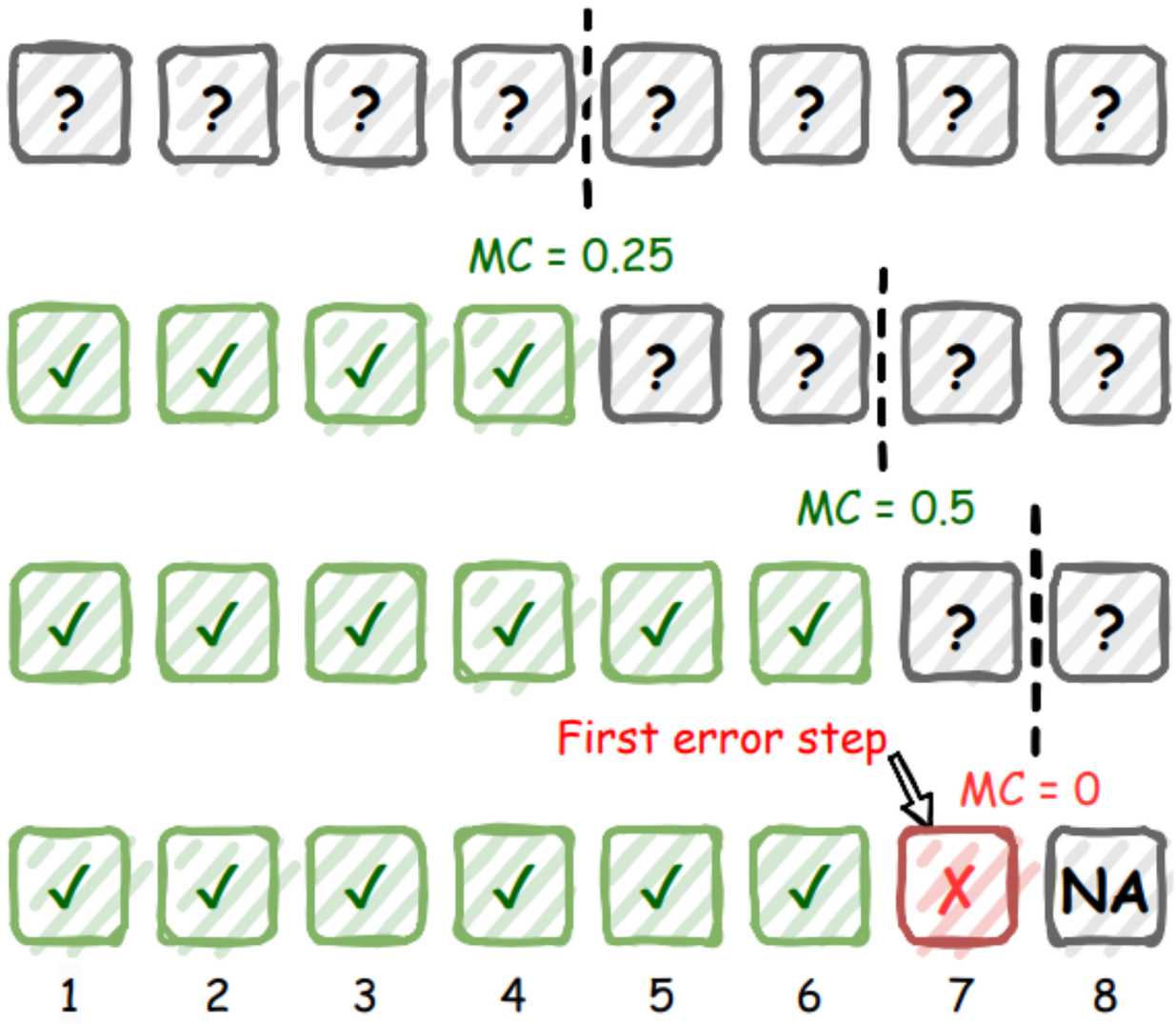
MC = 0.67

Monte Carlo Estimation using Binary Search

- To train a PRM, we need to locate the first error in a solution efficiently
- Given a solution with potential errors, split it at the midpoint m. We then perform rollouts for $s_{1:m}$ with two possible outcomes:
  - (1) $c_m > 0$, indicating that the first half of the solution is correct, as at least one correct answer can be rolled out from m-th step, and thus the **error is in the second half**;
  - (2) $c_m = 0$, indicating the **error is very likely in the first half**, as none of the rollouts from m-th step is correct.
- This process narrows down the error location to **either the first or second half of the solution**

Monte Carlo Estimation using Binary Search

- Locate the first error with a time complexity of $O(k \log M)$ rather than $O(kM)$ in the brute-force setting, where M is the total number of steps in the original solution
- Example:



Monte Carlo Tree Search

- **Goal**: collect multiple PRM training examples (a.k.a. triplets of question, partial solution, and correctness label) for a single question

- Brute-force way: independently sample multiple rollouts from scratch for a single question

- Efficient way: apply MCTS to store all rollouts during the sampling

Tree Structure

- Each tree node s contains the question q and prefix solution $x_{1:t}$, together with all previous rollouts $\{(s, r_i)\}^k_{i=1}$ from the state.

- Each edge (s, a) is either a single step or a sequence of consecutive steps from the node s.

- The nodes also store a set of statistics,

$$\{N(s), \text{MC}(s), Q(s, r)\},$$

where $N(s)$ denotes the visit count of a state, MC(s) represents the Monte Carlo estimation in Eq. (1), and $Q(s, r)$ is a state-rollout value function that is correlated to the chance of selecting a rollout during the selection phase of tree traversal

Selection

- Define the state-rollout value function

$$Q(s, r) = \alpha^{1-\mathrm{MC}(s)} \cdot \beta^{\frac{\mathrm{len}(r)}{L}},$$

where α, $\beta \in (0, 1]$ and $L > 0$ are constant hyperparameters; while len(r) denotes the length of a rollout in terms of number of tokens. A length penalty term penalize excessively long rollouts.

- **Key idea**: prioritize *supposed-to-be-correct* ***wrong-answers*** rollout during selection.
  - *supposed-to-be-correct*: high MC(s) closed to 1
  - ***wrong-answers***: a rollout has a wrong final answer

Selection

- Complete selection criteria, considering both **state-rollout values** and **tree nodes' visit counts**

$$(s, r) = \arg\max_{(s,r)} [Q(s, r) + U(s)]$$

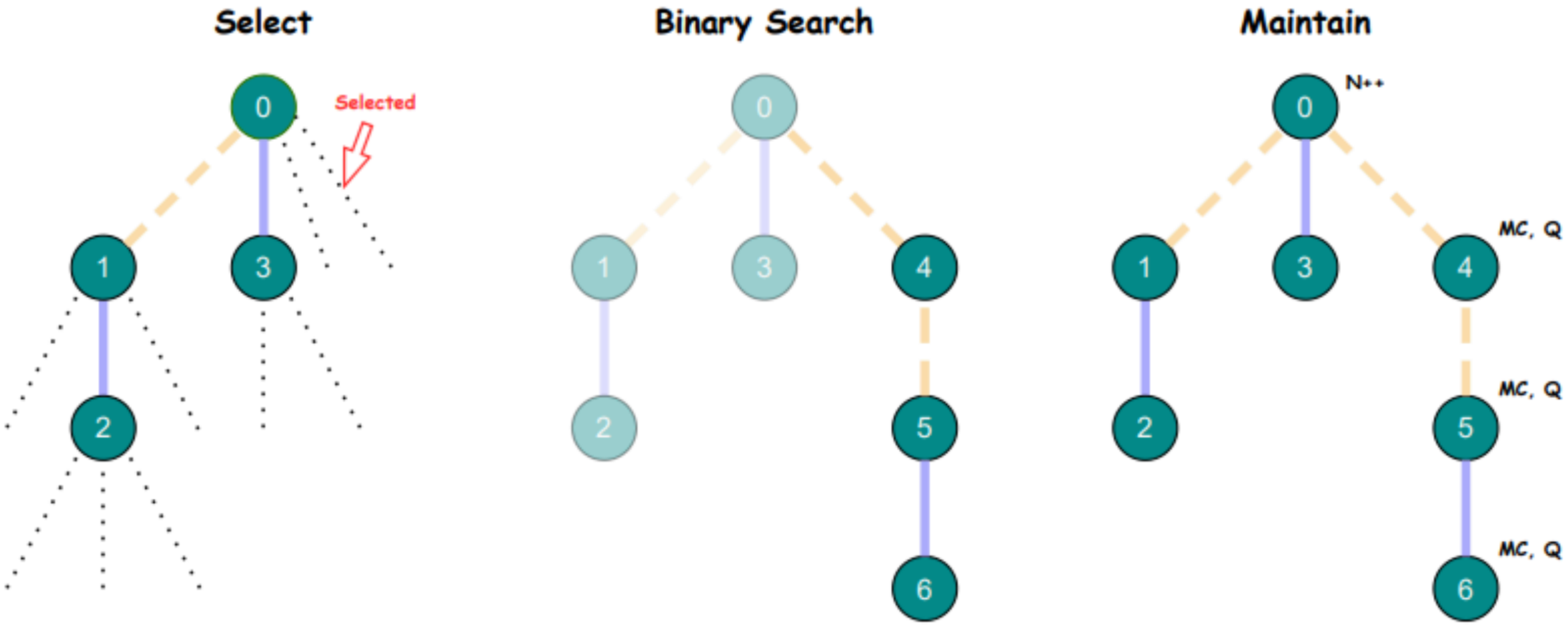
$$U(s) = c_{\text{puct}} \frac{\sqrt{\sum_i N(s_i)}}{1 + N(s)},$$

where c_{puct} is a constant determining the level of exploration.

This strategy initially favors rollouts with low visit counts but gradually favors those with high rollout value

Three Stages of MCTS

- Maintain stage: update N(s), and hence MC(s) and Q(s, r)



PRM Training

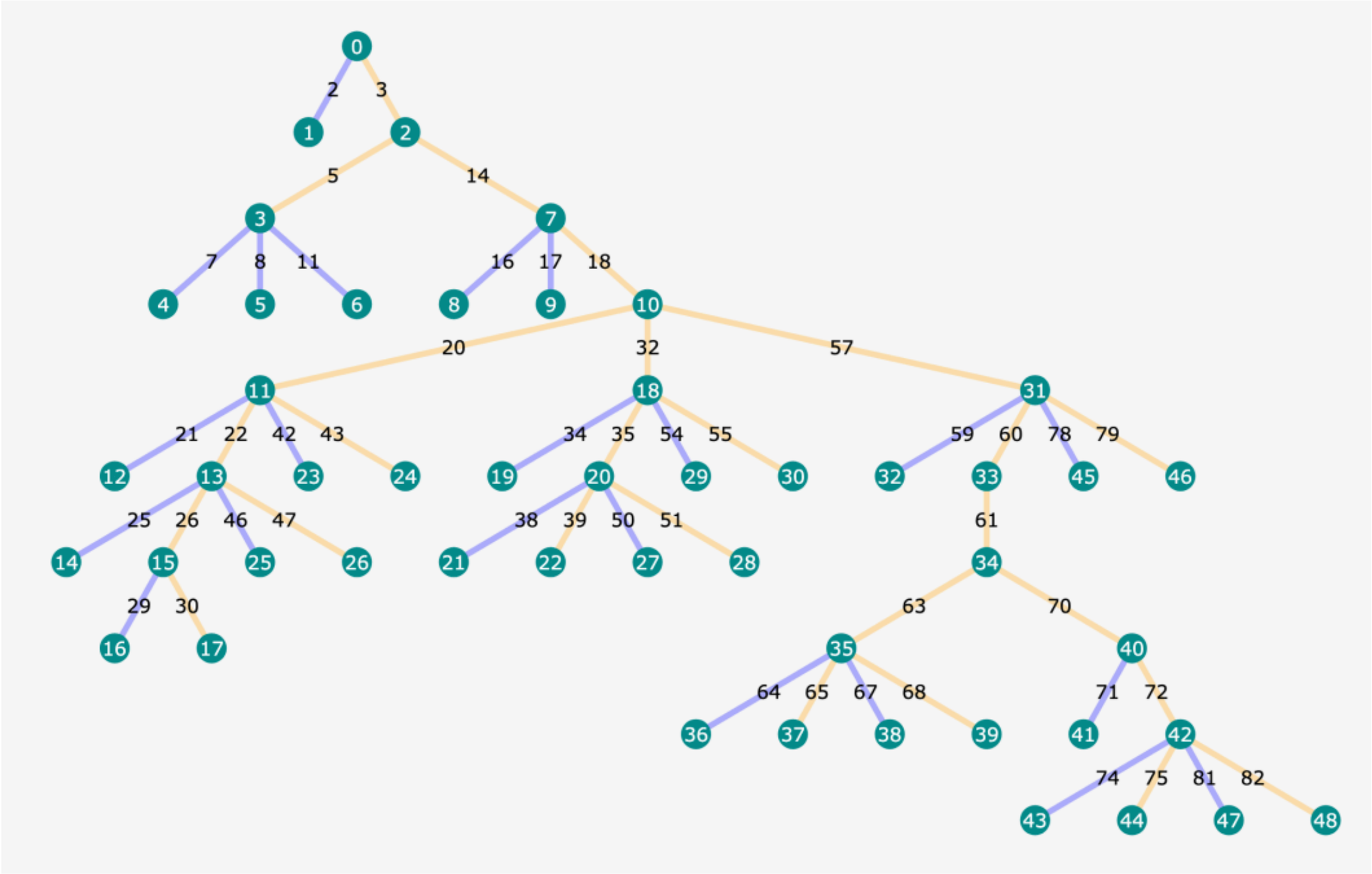
- After collecting edge (s, a) from the constructed state-action tree, we can trained a PRM using the standard classification loss.

$$\mathcal{L}_{\text{pointwise}} = \sum_{i=1}^{N} \hat{y}_i \log y_i + (1 - \hat{y}_i) \log(1 - y_i),$$

where $\hat{y}_i$ represents the correctness label and y_i = PRM(s, a) is the prediction score of the PRM

Example Tree Structure of OmegaPRM

- Yellow edges are correct steps and blue edges are wrong.



DeepSeek R1

- PRM has three main limitations
  - First, it is challenging to explicitly define a fine-grain step in general reasoning.
  - Second, determining whether the current intermediate step is correct is a challenging task. Automated annotation using models may not yield satisfactory results, while manual annotation is not conducive to scaling up.
  - Third, once a model-based PRM is introduced, it inevitably leads to reward hacking (Gao et al., 2022), and retraining the reward model needs additional training resources and it complicates the whole training pipeline.

DeepSeek R1

- Two limitations of MCTS when applying to LLMs
  - First, unlike chess, where the search space is relatively well-defined, token generation presents an exponentially larger search space. To address this, we set a maximum extension limit for each node, but this can lead to the model getting stuck in local optima.
  - Second, the value model directly influences the quality of generation since it guides each step of the search process. Training a fine-grained value model is inherently difficult, which makes it challenging for the model to iteratively improve. While AlphaGo's core success relied on training a value model to progressively enhance its performance, this principle proves difficult to replicate in our setup due to the complexities of token generation.

DeepSeek R1

- Use rule-based outcome rewards
  - **Accuracy rewards**: evaluates whether the answer 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**: enforces the model to put its thinking process between ‘<think>’ and ‘</think>’ tags.

DeepSeek R1

- Comparison of DeepSeek-R1-Zero and OpenAI o1 models on reasoning-related benchmarks.

Model	AIME 2024		MATH-500	GPQA Diamond	LiveCode Bench	CodeForces
	pass@1	cons@64	pass@1	pass@1	pass@1	rating
OpenAI-o1-mini	63.6	80.0	90.0	60.0	53.8	1820
OpenAI-o1-0912	74.4	83.3	94.8	77.3	63.4	1843
DeepSeek-R1-Zero	71.0	86.7	95.9	73.3	50.0	1444

Part 3: Reasoning without Verifiable Rewards

Big Problem for Reasoning (Revisited)

RLVR is great for math and code.

But how do you "verify" a...

...legal argument?

...medical diagnosis?

...business strategy?

...philosophical debate?

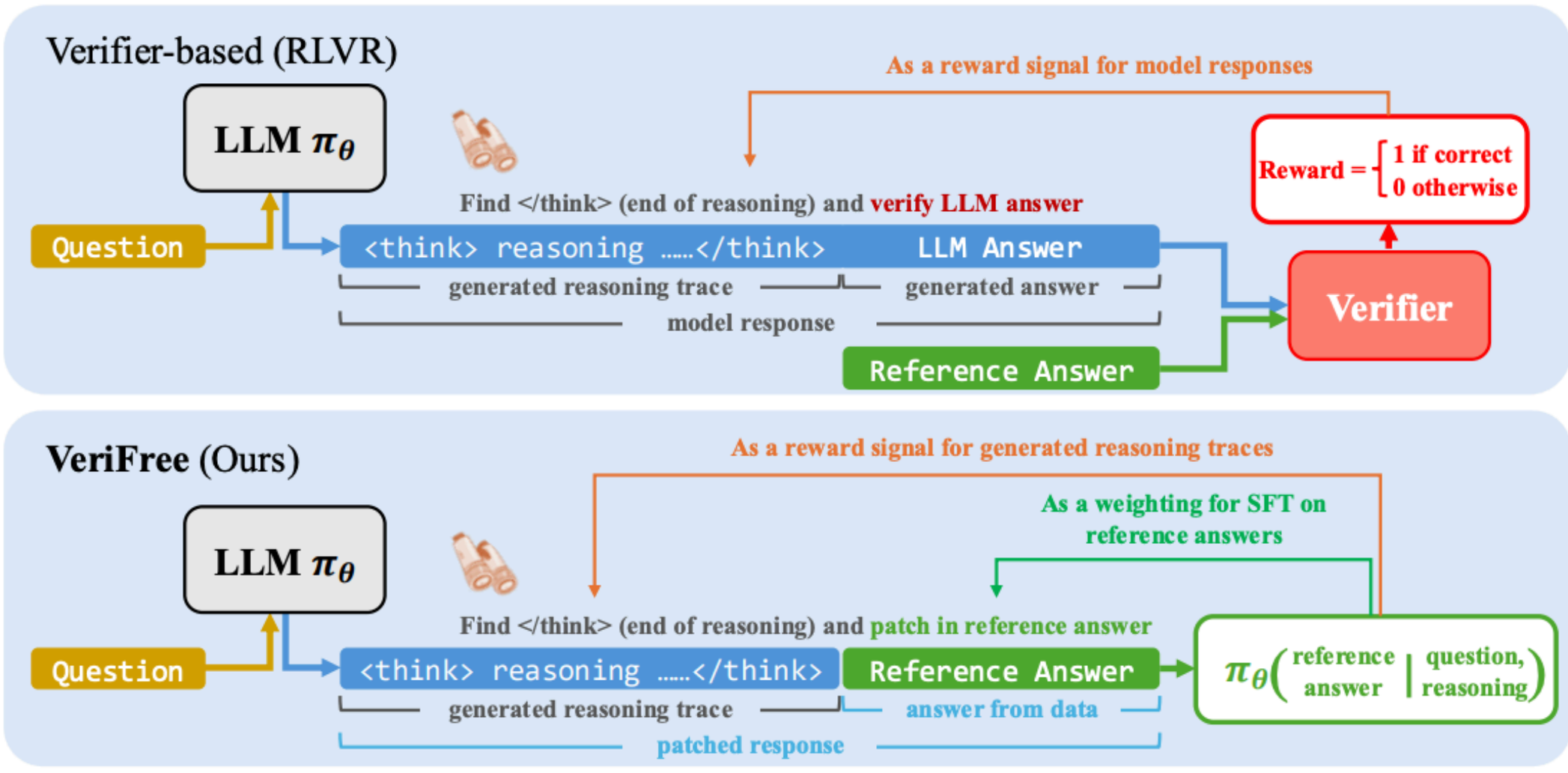
There is no "compiler" or "proof checker" for these tasks.

The "General Reasoning" Challenge

- **Goal**: Train models to reason well in ambiguous, open-ended domains.
- **Challenge**: How do we reward "good reasoning" when we can't automatically verify the final answer?

Reinforcing General Reasoning without Verifiers (Zhou et al. 2025)

- A "Verifier-Free" Method: This method bypasses answer verification entirely.



Reinforcing General Reasoning without Verifiers (Zhou et al. 2025)

- **VeriFree reward**: Use the policy model to evaluate how likely a generated reasoning trace can lead to the correct answer

Verifier-based (R1-Zero)	**VeriFree (Ours)**
Model generates the reasoning trace $\boldsymbol{z}$ and answer $\boldsymbol{y}$.	Model generates the reasoning trace $\boldsymbol{z}$.
Extract the answer $\boldsymbol{y}$.	Patch in the correct answer $\boldsymbol{y}^\star$.
Check answer using a verifier.	**Evaluate probability** $\pi_\theta(\boldsymbol{y}^\star \mid \boldsymbol{x}, \boldsymbol{z})$.
Reward $R_{\text{Verifier}} = 1$ if correct, 0 otherwise.	Reward $R_{\text{VeriFree}} = \pi_\theta(\boldsymbol{y}^\star \mid \boldsymbol{x}, \boldsymbol{z})$.
Train with gradient estimator $\nabla_\theta J_{\text{Verifier}}$ (Eq. 3).	Train with gradient estimator $\nabla_\theta J_{\text{VeriFree}}$ (Eq. 5).

Figure 3: A pseudocode-like comparison of VeriFree (ours) and the standard R1-Zero approach.

$$\nabla_\theta J_{\text{VeriFree}}(\theta; \boldsymbol{x}, \boldsymbol{y}^\star) = \mathbb{E}_{\boldsymbol{z}\sim\pi_\theta(\cdot|\boldsymbol{x})}\Big[R_{\text{VeriFree}}(\boldsymbol{x}, \boldsymbol{y}^\star, \boldsymbol{z})\big[\underbrace{\nabla_\theta \log \pi_\theta(\boldsymbol{z}|\boldsymbol{x})}_{\text{reasoning term}} + \underbrace{\nabla_\theta \log \pi_\theta(\boldsymbol{y}^\star|\boldsymbol{x}, \boldsymbol{z})}_{\text{reference answer term}}\big]\Big].$$

VeriFree Analogy

Imagine a student shows their math work.

Instead of grading the final answer, the teacher (the reward) says:

"Given the steps you've written, how confident are you that the answer is X?"

(Where X is the known correct answer from the textbook.)

VeriFree Result

- It works! It matches or even outperforms expensive verifier-based methods on general reasoning benchmarks like GPQA and MMLU-Pro.

- Key Takeaway: We can train general reasoning without a verifier, making it:
  - Simpler (no separate RM)
  - Faster to train
  - Less memory-intensive
  - More robust to reward hacking

Comparison with Recent Works on RL without Verifiable Rewards

- Reuse the policy model in different ways to judge the quality of a reasoning trace

$$\nabla_\theta J_{\text{Verifier}} = \mathbb{E}_{\boldsymbol{z},\boldsymbol{y}}\Big[\, \mathbb{1}_{\{\boldsymbol{y}\equiv\boldsymbol{y}^\star\}} \overbrace{\nabla_\theta \log \pi_\theta(\boldsymbol{z}|\boldsymbol{x})}^{\text{reasoning term}} + \mathbb{1}_{\{\boldsymbol{y}\equiv\boldsymbol{y}^\star\}} \overbrace{\nabla_\theta \log \pi_\theta(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{z})}^{\text{answer term}} \Big] \qquad \text{(R1-Zero)}$$

$$\nabla_\theta J_{\text{VeriFree}} = \mathbb{E}_{\boldsymbol{z}}\Big[\, \pi_\theta(\boldsymbol{y}^\star|\boldsymbol{x},\boldsymbol{z})\, \nabla_\theta \log \pi_\theta(\boldsymbol{z}|\boldsymbol{x}) + \pi_\theta(\boldsymbol{y}^\star|\boldsymbol{x},\boldsymbol{z}) \overbrace{\nabla_\theta \log \pi_\theta(\boldsymbol{y}^\star|\boldsymbol{x},\boldsymbol{z})}^{\text{reference answer term}} \Big] \qquad \text{(Ours)}$$

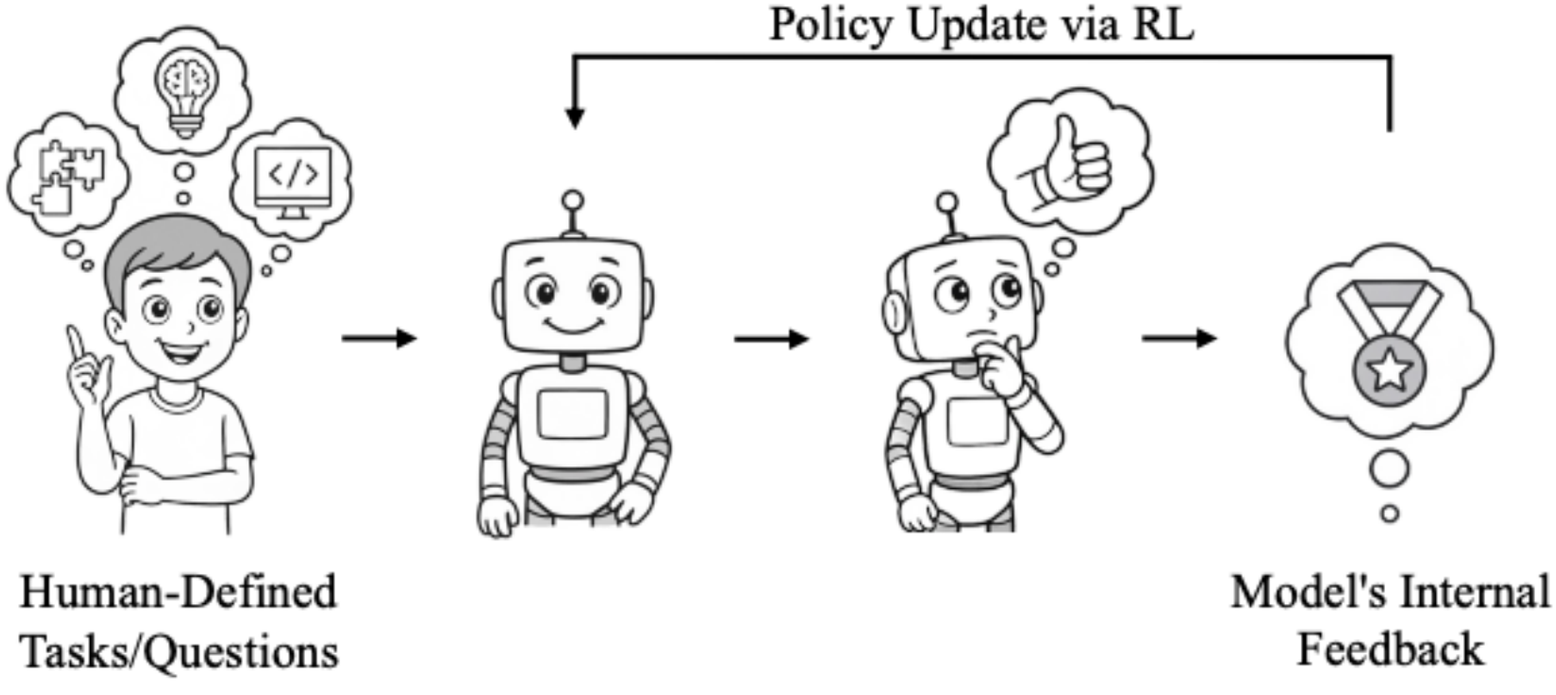
$$\nabla_\theta J_{\text{JLB}} = \mathbb{E}_{\boldsymbol{z}}\Big[\, \log \pi_\theta(\boldsymbol{y}^\star|\boldsymbol{x},\boldsymbol{z})\, \nabla_\theta \log \pi_\theta(\boldsymbol{z}|\boldsymbol{x}) + 1 \cdot \nabla_\theta \log \pi_\theta(\boldsymbol{y}^\star|\boldsymbol{x},\boldsymbol{z}) \Big] \qquad \text{(Tang et al. [40])}$$

$$\nabla_\theta J_{\text{LaTRO}} = \mathbb{E}_{\boldsymbol{z}}\Big[\, \big(\log \pi_\theta(\boldsymbol{y}^\star|\boldsymbol{x},\boldsymbol{z}) - \log \tfrac{\pi_\theta(\boldsymbol{z}|\boldsymbol{x})}{\pi_{\text{ref}}(\boldsymbol{z}|\boldsymbol{x})}\big)\, \nabla_\theta \log \pi_\theta(\boldsymbol{z}|\boldsymbol{x}) + 1 \cdot \nabla_\theta \log \pi_\theta(\boldsymbol{y}^\star|\boldsymbol{x},\boldsymbol{z}) \Big]$$

(Chen et al. [4])

Learning to Reason without External Rewards (Zhao et al. 2025)

- LLMs learn from intrinsic signals generated by the model itself, without external supervision

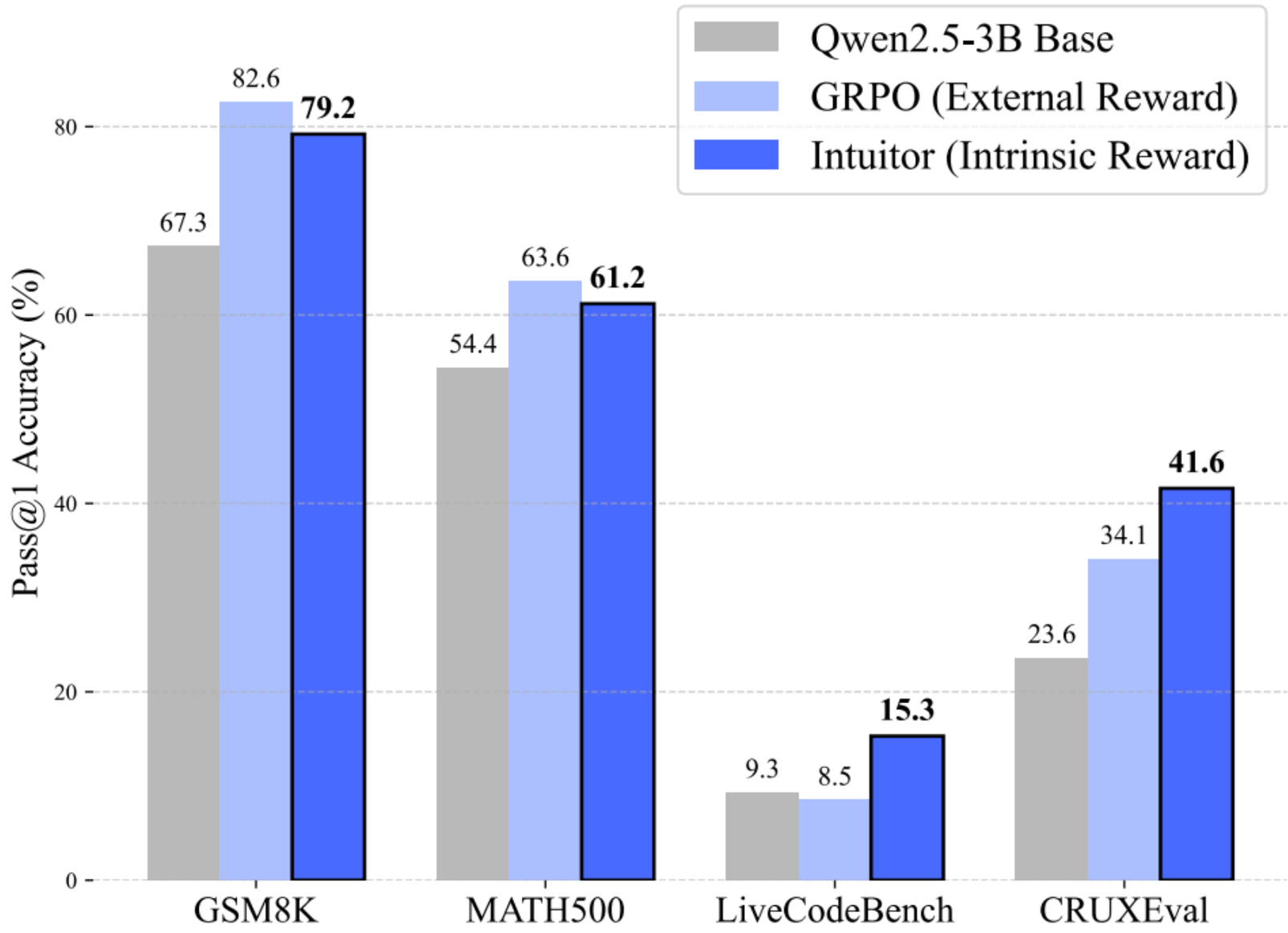


Self-certainty Reward

- Adopt the self-certainty metric from Kang et al. [2025], defined as the **average KL divergence between a uniform distribution U over the vocabulary V and the model's next-token distribution**:

$$\textbf{Self-certainty}(o|q) := \frac{1}{|o|}\sum_{i=1}^{|o|}\mathrm{KL}(U \parallel p_{\pi_\theta}(\cdot|q, o_{<i})) = -\frac{1}{|o|\cdot|\mathcal{V}|}\sum_{i=1}^{|o|}\sum_{j=1}^{|\mathcal{V}|}\log\left(|\mathcal{V}|\cdot p_{\pi_\theta}(j|q, o_{<i})\right)$$

Self-certainty Results



Summary

- **1. Outcome Supervision (OS):** Rewarded the final answer. (Brittle, "lucky guesses")
- **2. Process Supervision (PS):** Rewarded every step. (Robust, but expensive)
  (Paper: "Let's verify step by step")
- **3. Automated PS:** Scaled process supervision.
  (Paper: "Automated Process Supervision")
- **4. Verifiable Rewards (RLVR):** Used automated outcome checks for formal domains.
- **5. Verifier-Free & Internal Feedback:** The frontier. Using likelihood or "self-certainty" as a reward for general, ambiguous reasoning.
  (Papers: "VeriFree" & "Intuitor")

Questions?