

# FALL 2024 COS597R: DEEP DIVE INTO LARGE LANGUAGE MODELS

Danqi Chen, Sanjeev Arora



Lecture 14: LLM reasoning + Role of inference-time compute

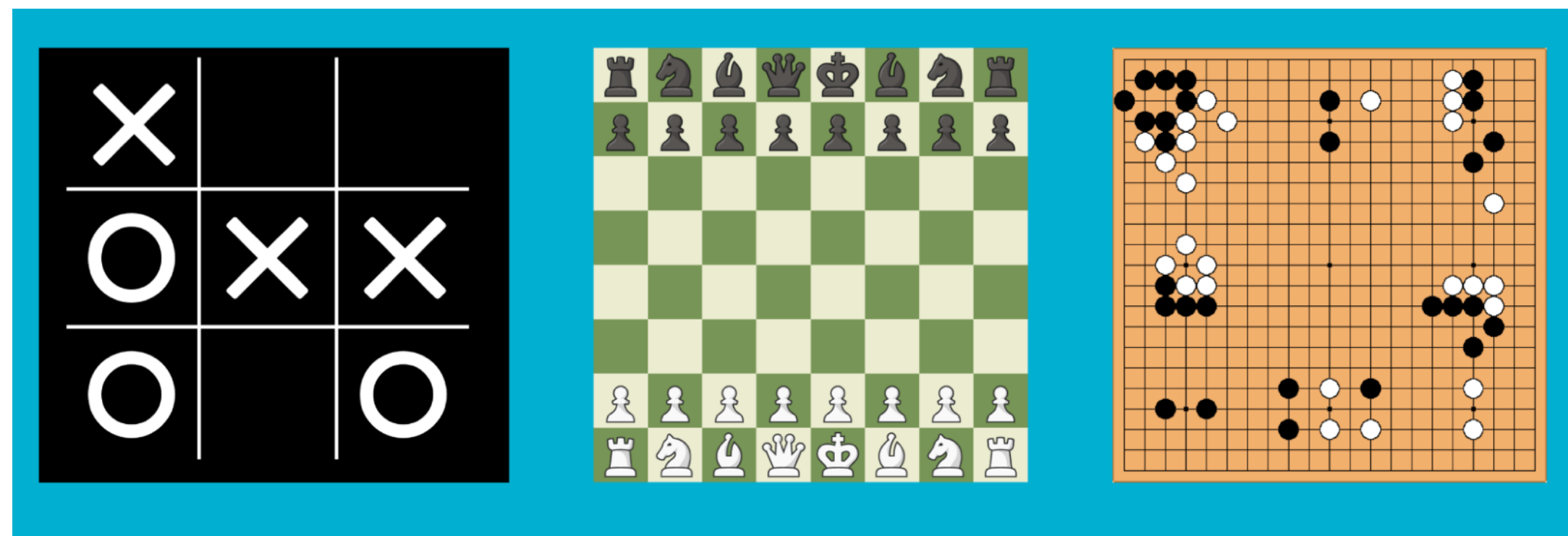
<https://princeton-cos597r.github.io/>

# Reasoning is hard

(e.g., science/math questions; research problems; Chess/Go)

Scientists come up with many wrong solutions before hitting upon the right one

Chess/Go players contemplate many moves, looking ahead many steps



**“Inference time compute”**

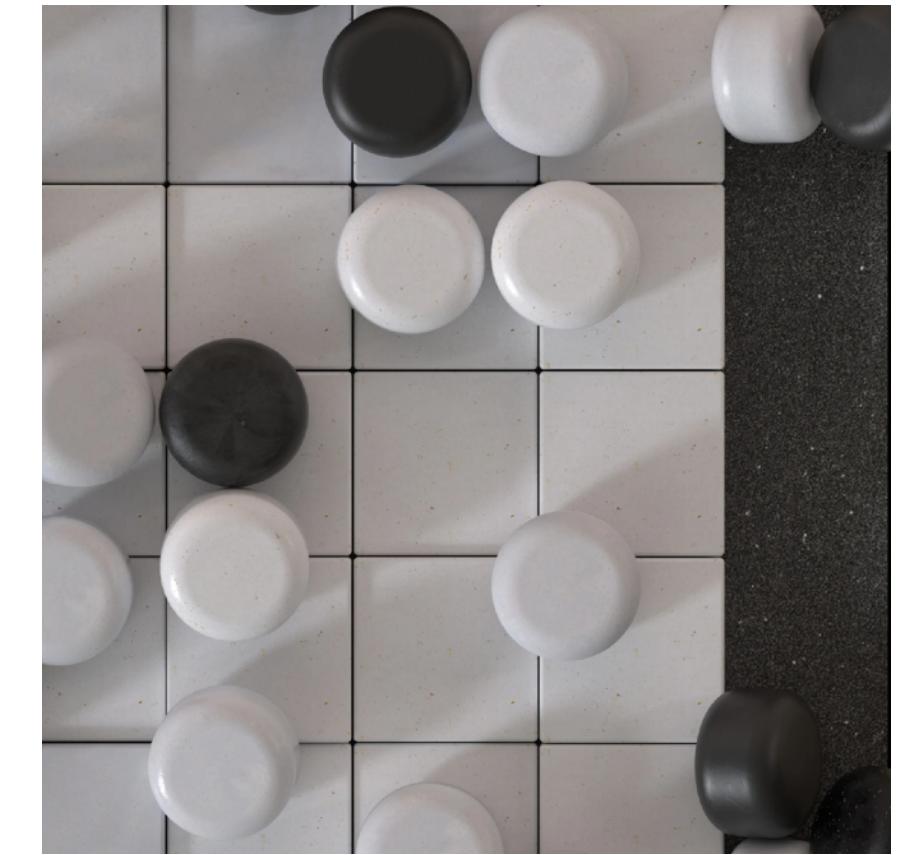
Can we enhance LLM reasoning by allowing them more compute steps at inference time?

---

# Background: RL ideas from Game Playing

(And how to think about language generation/reasoning as a policy)

# Learning to play Go

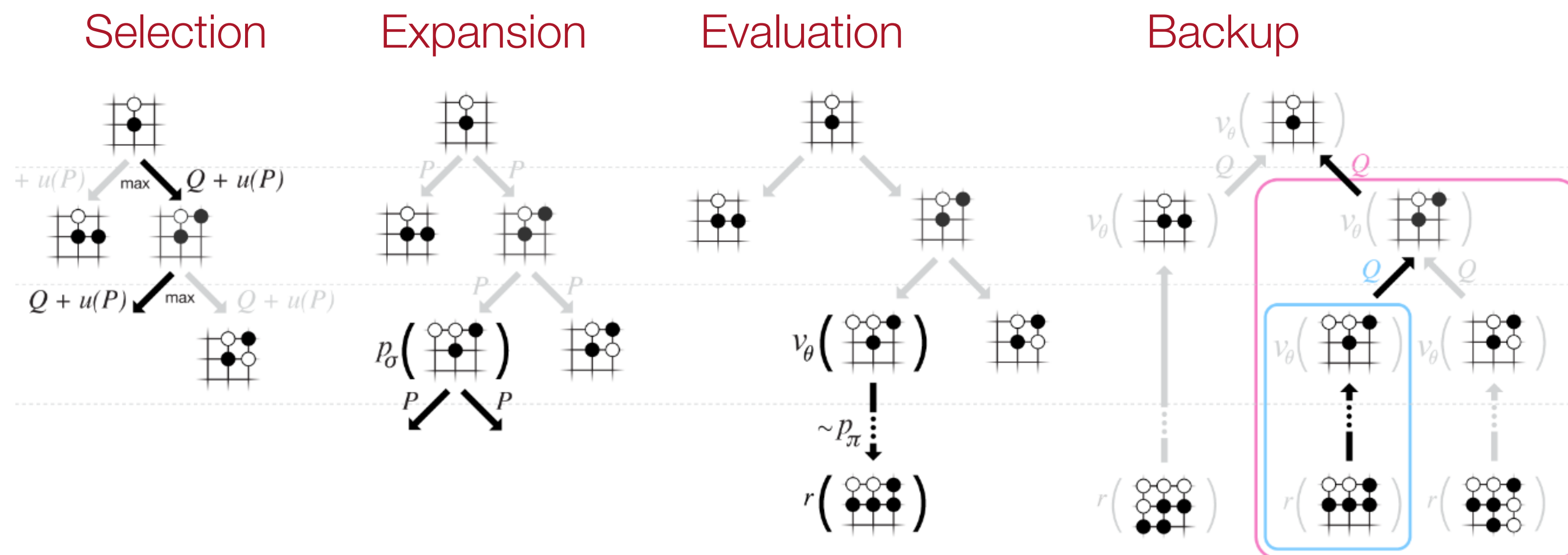


Elo-rating of World champion: 3600

Vanilla AlphaGo's rating (deep net to represent "policy function", i.e. next-move) **3000**

Alphago rating with "inference-time compute": **5000**

Monte-Carlo Tree Search: many "policy rollouts" and then use "value function" to select best next move. (Policy and value functions represented by two deep nets)



Why this works: Local **inconsistencies** in policy and value function get ironed out via inference time roll-outs

# Monte-Carlo Tree Search (MCTS)

(named by Remi Coulom 2006)

## Wikipedia summary

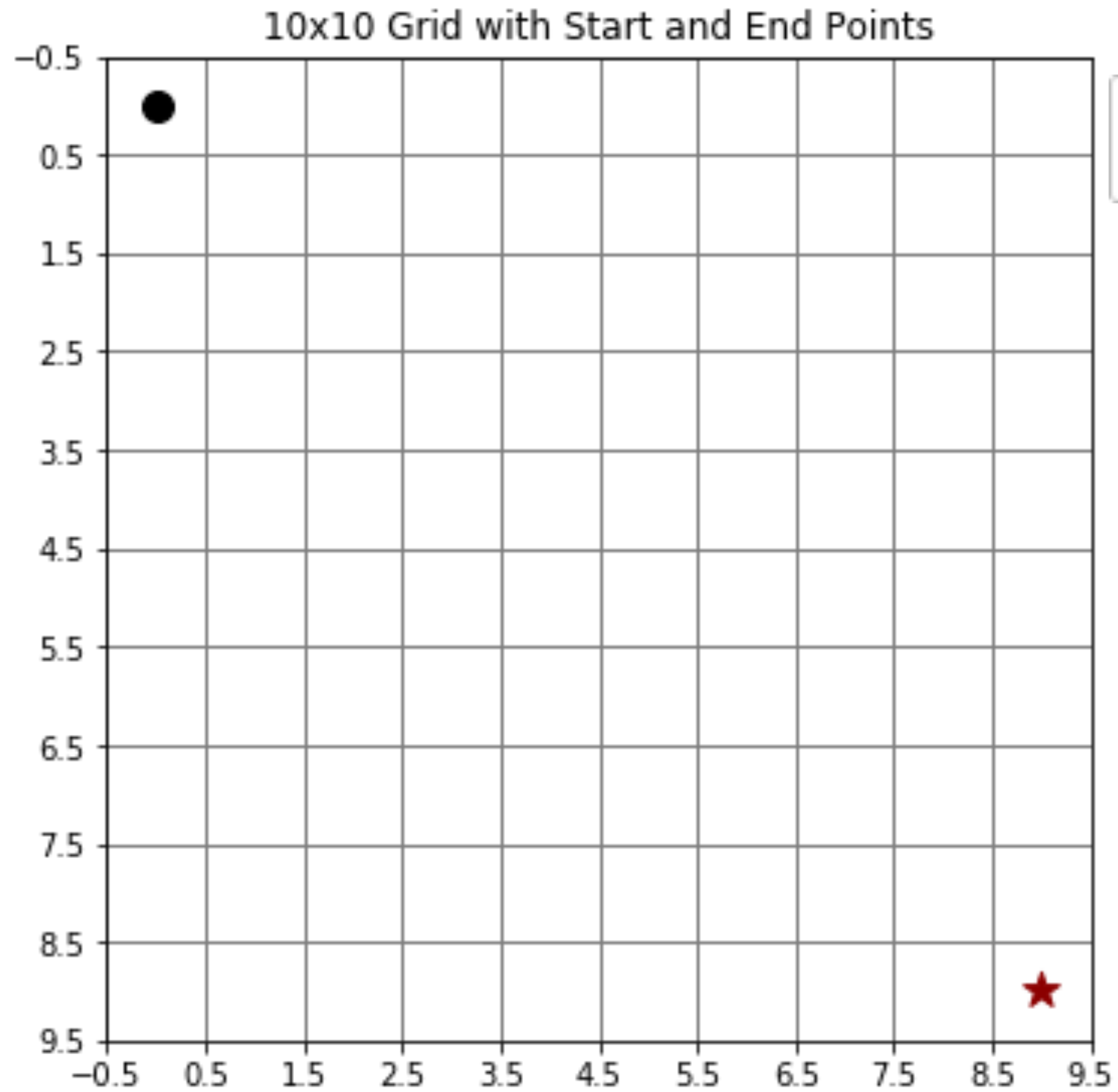
The focus of MCTS is on the analysis of the **most promising moves**, expanding the search tree based on **random sampling** of the search space. The application of Monte Carlo tree search in games is based on many playouts, also called **roll-outs**. In each playout, the game is played out to the very end by selecting moves at random. The final game result of each playout is then used to **weight the nodes** in the game tree so that better nodes are more likely to be chosen in future playouts.

# Important caveat

Go-playing uses MCTS also to **improve** the policy and value function

Today we're just considering MCTS ideas at inference time

# Illustration



- Start (Dark Dot)
- ★ End (Dark Star)

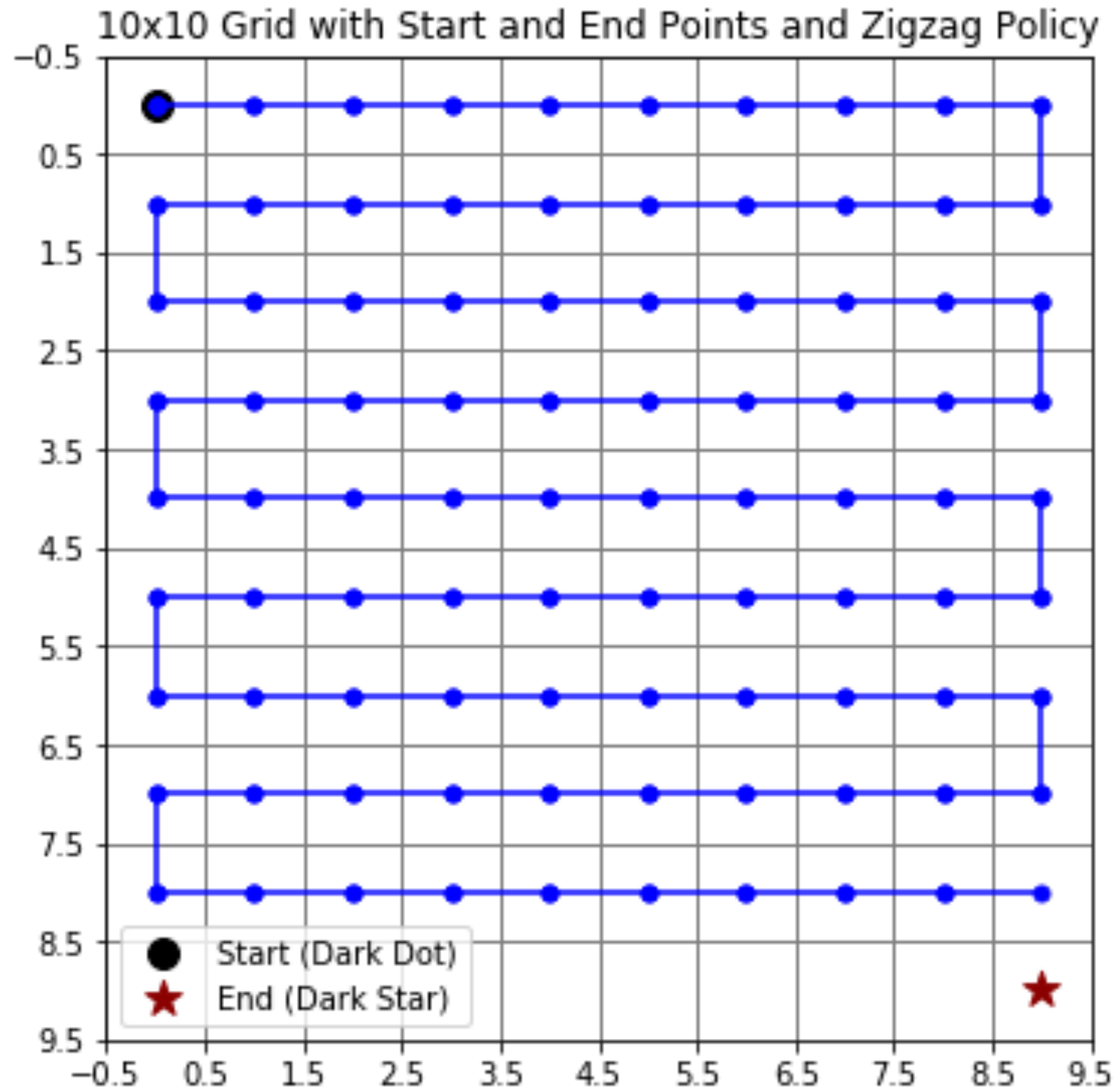
Allowed moves: one step in any direction including diagonal

## Value function:

— (Euclidean distance to End point)

(In general, value function needs to be **learned** )

# Current value function and policy



## Value function:

– (Euclidean distance to end point)

At the start the value is

$$-\sqrt{(9 - 0)^2 + (9 - 0)^2} = \sqrt{162} = -12.73$$

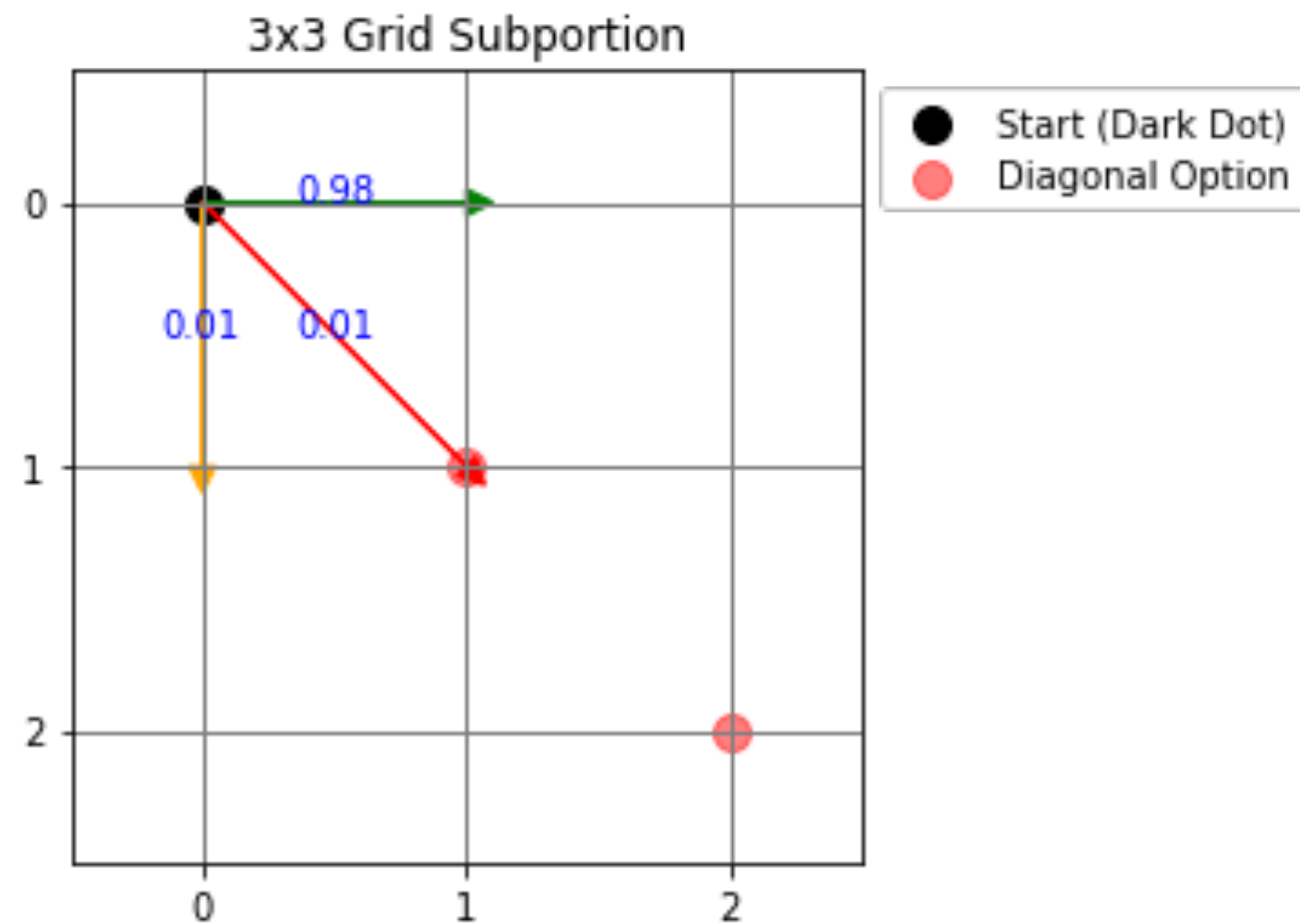
(In complicated settings, value function will need to be learned)

## Current Policy:

- Blue edges have probability 0.98
- All other allowed moves have small nonzero probability



# Why MCTS can improve over policy



Policy rollout (i.e., lookahead with 3 moves) reveals that diagonal move gives best improvement in value according to current value function

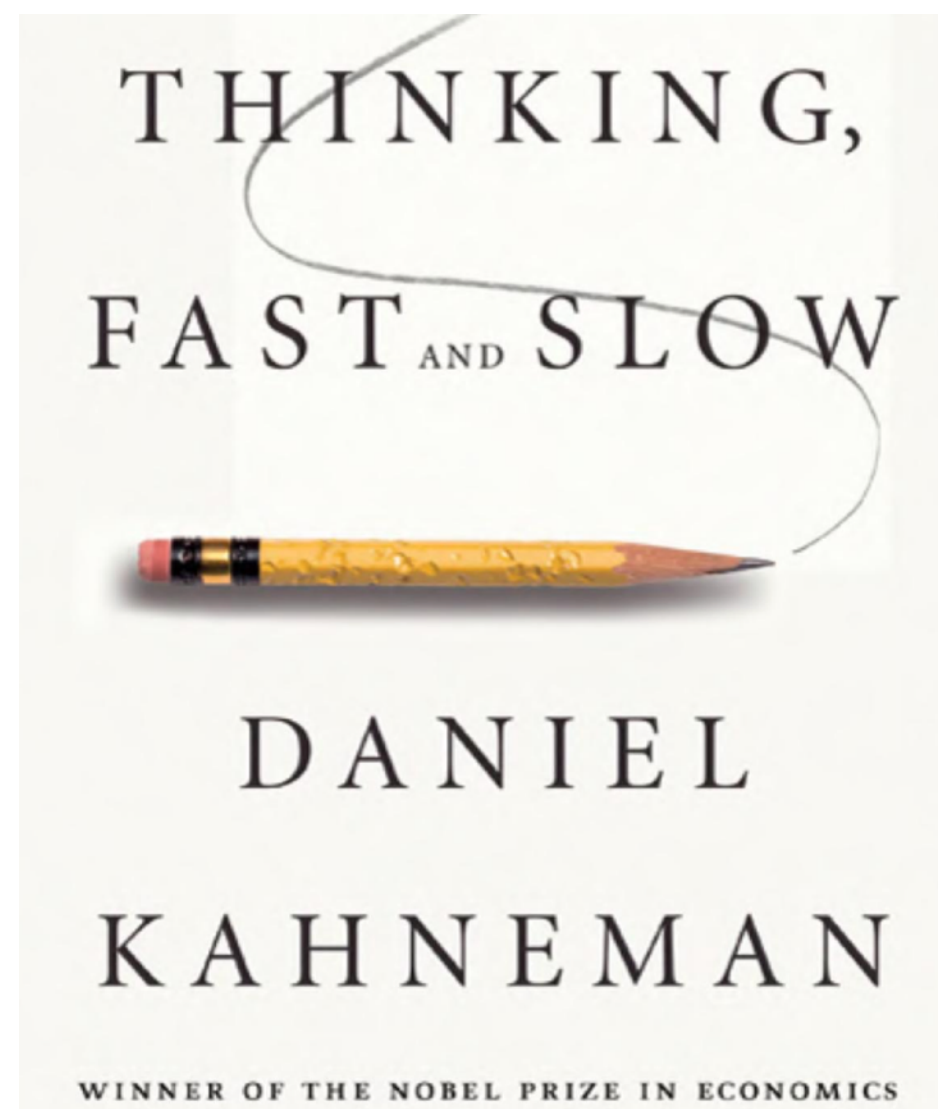
Takeaway: It suffices to learn a policy “in the right ballpark” of optimum policy, and then use search to improve moves at run-time

Some thought shows that limited lookahead cannot **guarantee** optimum move in general, since that may require looking ahead **a lot** of moves

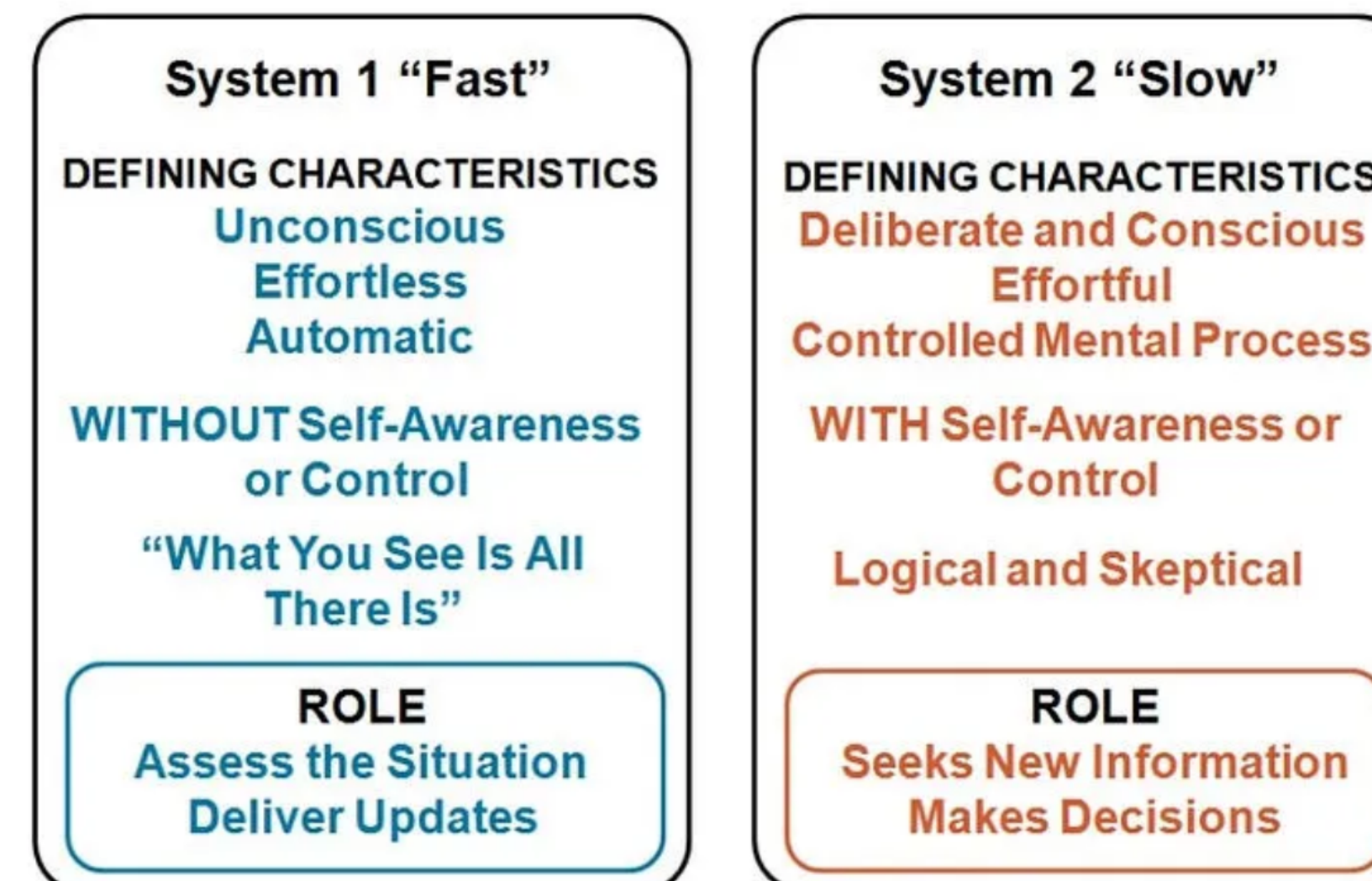
Pretraining budget of frontier LLMs = \$100M

Inference cost = \$0.002 cents/token

**Can we use more inference-time compute to enable better LLM reasoning?**

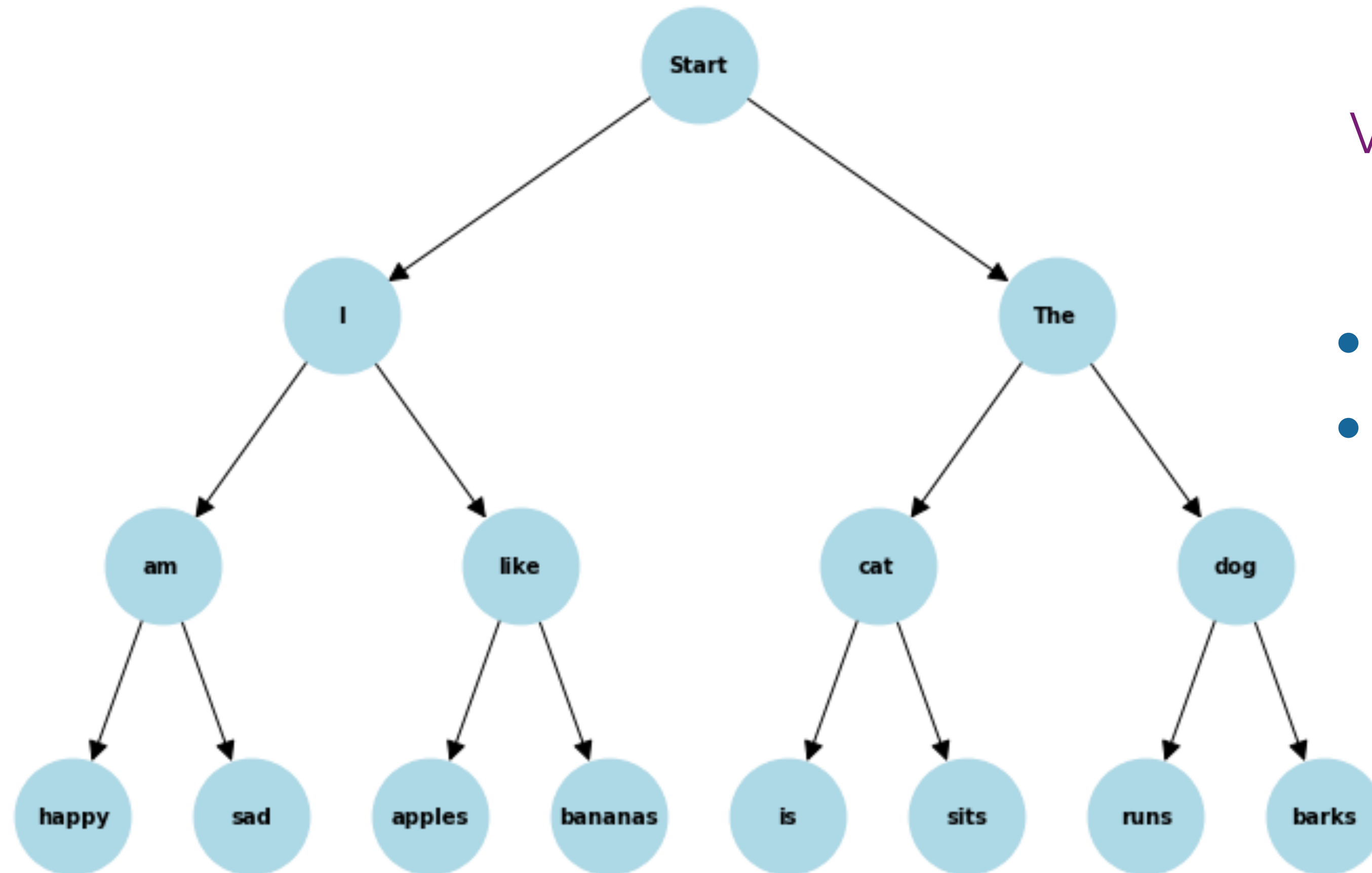


## System 1 and 2 thinking



# LLM Generation = “Policy”

LLM as Next-Word Policy Generator



Value function = “quality” of response

- How do we learn a good value function?
- Can we use policy rollouts at inference time to improve value of the generated response?

# Pass@k metric

Pass@k

“Generate k answers to the query; pick the best.”

(e.g., Pass@5 often much better than Pass@1 )

**PassRatio@K (when answer is a single word or number):**

**Generate K answers and determine the fraction that are correct**

Evaluating “Best” requires a reward Model for correctness/quality of math Q&A

**Qs: What is an obvious idea for learning a “reward” function for math?**

# Pass@k viewed as inference-time search

Pass@k

“Generate k answers to the query; pick the best.”

(e.g., Pass@5 often much better than Pass@1 )

Evaluating “Best” requires a reward Model for correctness/quality of math Q&A

**PassRatio@K (when answer is a single word or number):**

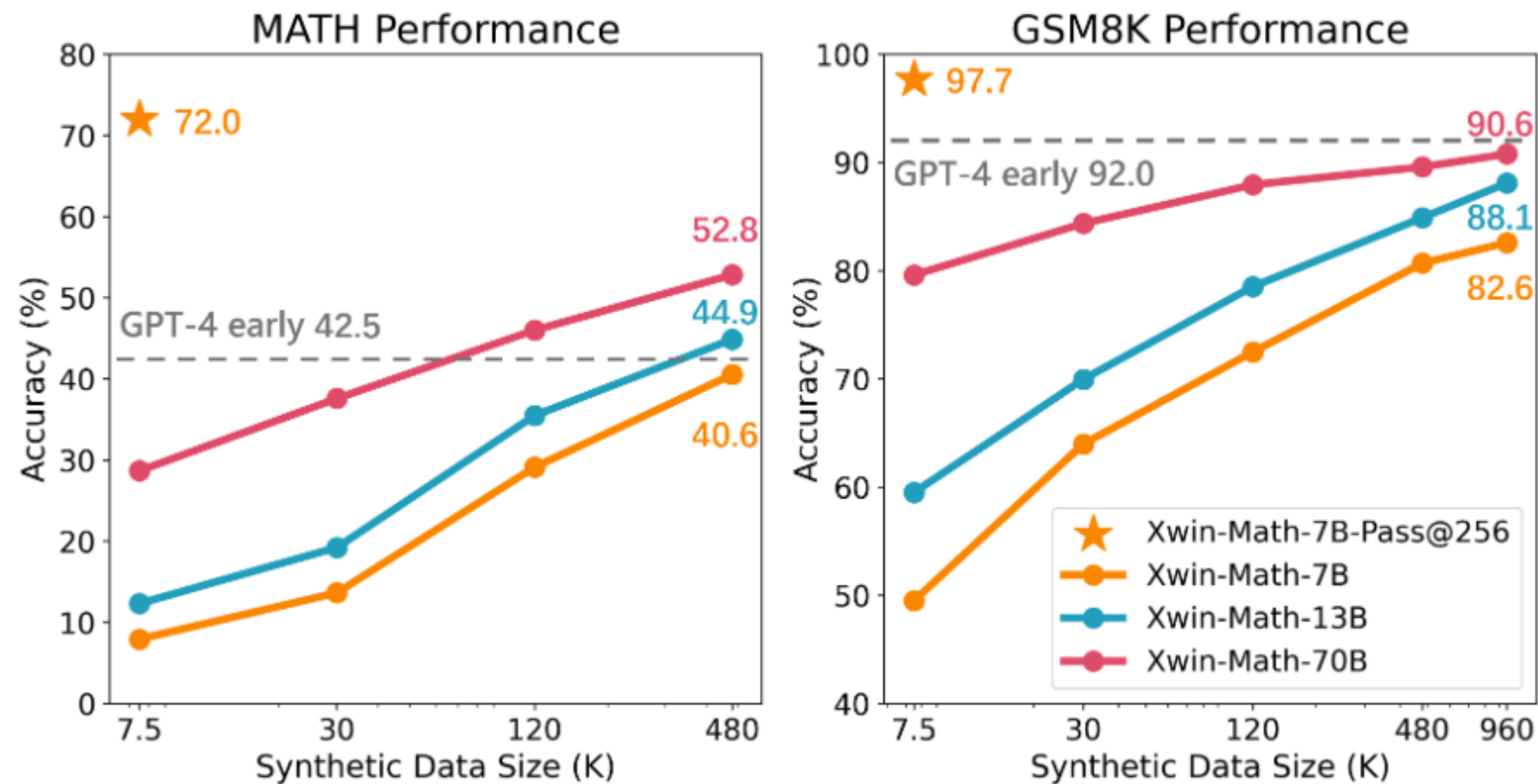
**Generate K answers and determine the fraction that are correct**

**Hurdle in learning reward function using naive SFT on human-labeled examples:**

- Even strong models can't reliably tell good answers from wrong
- Using wrong labels in SFT hurts performance super proportionately

**Today's examples (GSM8K and MATH) have autoverifiable answers**

# Potential power of best-of-k for math



Best-of-256 using Llama2 7B beats GPT4 (early version) and custom math models

PassRatio@256 much worse: 49% on GSM8K, 8% on MATH

Figure 1: The orange star markers represent the accuracy achieved by selecting the best response from 256 random generations of the LLaMA-2 7B model. The high accuracy on the MATH (left) and GSM8K (right) benchmarks (72.0% and 97.7%, respectively) suggest that the LLaMA-2 7B already possesses strong mathematical capabilities, although the stability in generating correct answers could be enhanced. This paper

$\text{Pass@K} - \text{PassRatio@K} = \text{Theoretical improvement from perfect reward model}$

Common 7B Language Models Already Possess Strong Math Capabilities

Chen Li<sup>1,4</sup>, Weiqi Wang<sup>2,4</sup>, Jingcheng Hu<sup>3,4</sup>, Yixuan Wei<sup>3,4</sup>, Nanning Zheng<sup>1</sup>, Han Hu<sup>4</sup>, Zheng Zhang<sup>4\*</sup>, Houwen Peng<sup>4\*</sup>

---

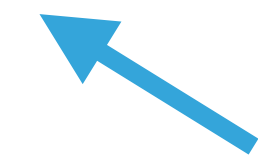
# Training Reward Model (RM) for math

Let's Verify Step by Step

**Hunter Lightman\***   **Vineet Kosaraju\***   **Yura Burda\***   **Harri Edwards**  
**Bowen Baker**   **Teddy Lee**   **Jan Leike**   **John Schulman**   **Ilya Sutskever**

# Outcome Supervision (ORM)

Given: GSM8K question, CoT answer, Final Answer.



Just a number

Goal: Train RM to say whether final answer is correct

Training data: usual +ve and -ve examples

Note: We saw ORM implicitly in earlier settings such as RLHF (i.e., preference pairs)






# Process Supervision (PRM)

Given: GSM8K question, CoT answer, Final Answer




Goal: Give a reward 0/ 1/ -1 to each step i.e., (neutral/positive/negative)

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?

   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  $x = 7$ .

PRM800K: Dataset contains 800K step-level labels provided by human raters across 75K solutions to 12K problems (MATH 8K training set + 4K test questions).

# PRK800K: Design Decisions

**“Convincing solutions”**: Those rated highly by our current best PRM

**“Wrong-answer”** : solutions that reach an incorrect final answer.

(note: final answer is a number or an expression; hence auto-checkable)

PRK800 uses Convincing Wrong answers (Generate N solutions per question, and choose K wrong-answer solutions that were most convincing to human raters)

**“Process RM” (PRM) training**: Train to predict (via log-likelihood loss)

the human annotation for each step

When we provide process supervision, we deliberately choose to supervise only up to the first incorrect step. This makes the comparison between outcome and process supervision more straightforward. For correct solutions, both methods provide the same information, namely that every step is correct. For incorrect solutions, both methods reveal the existence of at least one mistake, and process supervision additionally reveals the precise location of that mistake.

# Example of answer labeled by 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  $p_1(1) + p_2(1) + \cdots + p_k(1)$ .

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:  $0$

# ORM vs PRM Training

(Text from paper)

We train the large-scale PRM using the step-level labels in PRM800K. To ensure the large-scale ORM baseline is as strong as possible, we train on 100 uniform samples per problem from the generator.

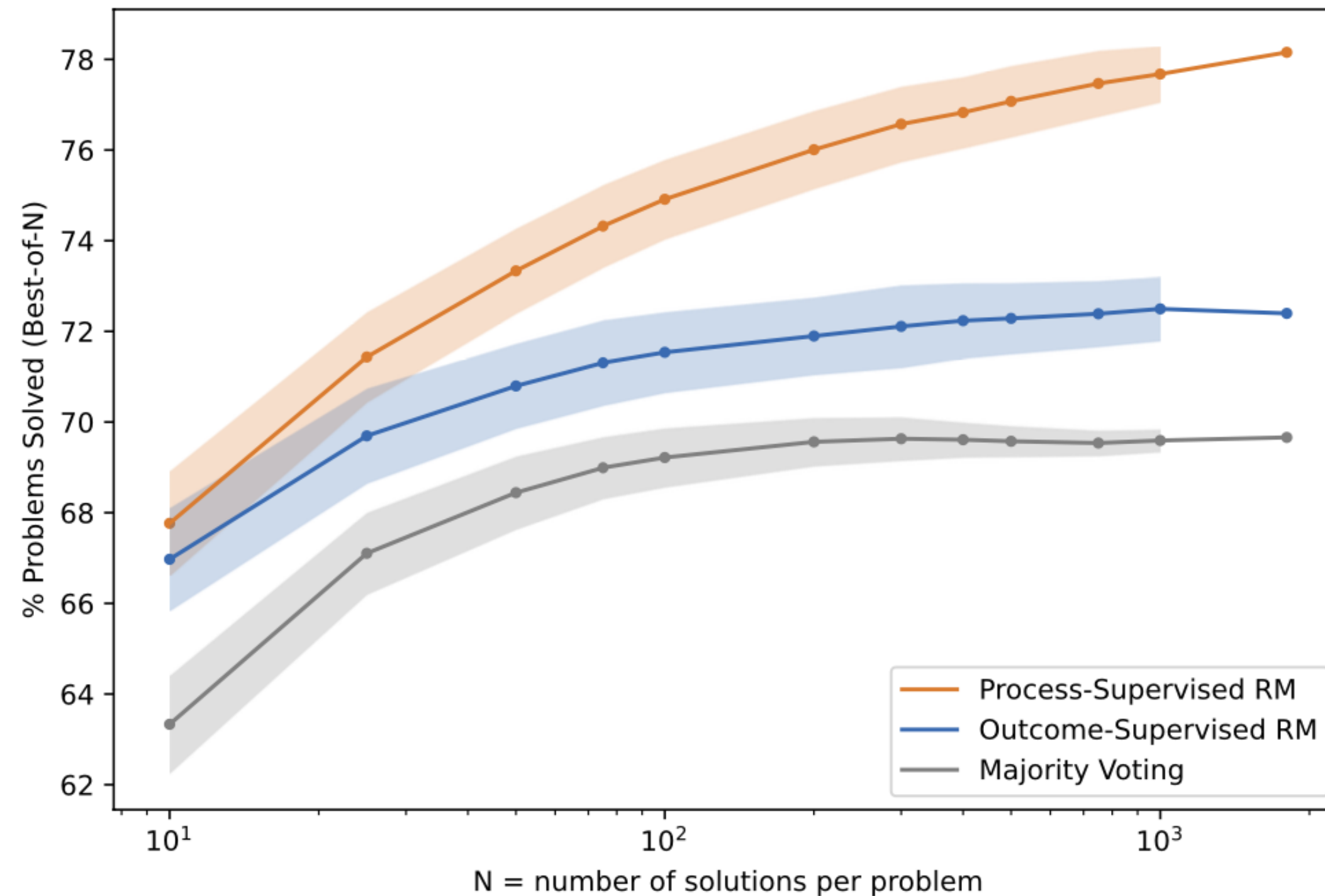
This means the ORM training set has no overlap with PRM800K, and it is an order of magnitude larger.

Although these two training sets are not directly comparable, each represents our best attempt to advance the state-of-the-art with each form of supervision. We note that training the ORM solely on PRM800K solutions would be problematic, since our active learning strategy has heavily biased the dataset towards wrong-answer solutions.

Discussion: What do you think could be a significant issue with such trained ORM/PRMs?

# Results

	ORM	PRM	Majority Voting
% Solved (Best-of-1860)	72.4	<b>78.2</b>	69.6



Note: Gap widens with more # of solutions to evaluate (PRM's effectiveness as helper improves at scale)

Figure 3: A comparison of outcome-supervised and process-supervised reward models, evaluated by their ability to search over many test solutions. Majority voting is shown as a strong baseline. For  $N \leq 1000$ , we visualize the variance across many subsamples of the 1860 solutions we generated in total per problem.

# OOD Generalization

ORM and PRM on a held-out set of 224 STEM questions, pulled from the most recent AP Physics, AP Calculus, AP Chemistry, AMC10, and AMC12 exams.

	ORM	PRM	Majority Vote	# Problems
AP Calculus	68.9%	<b>86.7%</b>	80.0%	45
AP Chemistry	68.9%	<b>80.0%</b>	71.7%	60
AP Physics	77.8%	<b>86.7%</b>	82.2%	45
AMC10/12	49.1%	<b>53.2%</b>	32.8%	84
Aggregate	63.8%	<b>72.9%</b>	61.3%	234

Table 1: We measure out-of-distribution generalization using recent STEM tests. We evaluate the outcome-supervised RM, the process-supervised RM, and majority voting using 100 test samples per problem.

# Benefits of Process Supervision

Process supervision is more likely to produce interpretable reasoning, since it encourages models to follow a process endorsed by humans. Process supervision is also inherently safer: it directly rewards an aligned chain- of-thought rather than relying on outcomes as a proxy for aligned behavior

This also seems to **discourage** reward hacking

Unlike many other settings, there is **no alignment tax**. In fact, an alignment benefit!



# Discussion

What more would you have liked the OpenAI team to report in the paper?

---

Using RM's to improve math reasoning

# Best-of-N SFT

R: reward model (ORM or PRM) for the dataset, e.g., GSM8K

**M**: Model to be improved

Method:

Generate many solutions for each question using M

Filter using RM to get a (likely) good solution for each question.

SFT on good (question, solution) pairs

# SFT performance (y-axis is log of # params)

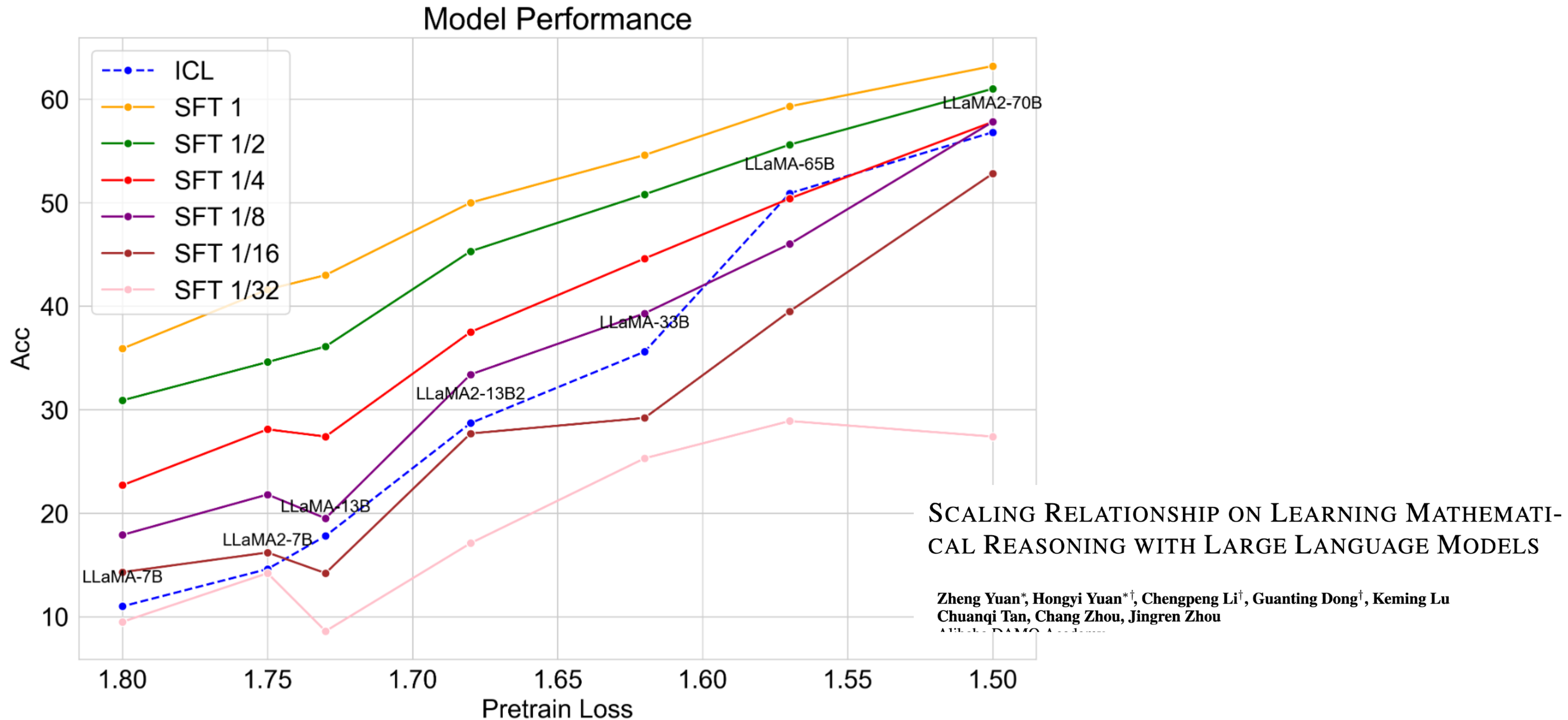


Figure 3: The performance of SFT with different amounts of supervised data on GSM8K.

# Rejection Fine Tuning (RFT)

Similar to SFT, except use **many** models to generate solutions, and use all solutions OK'ed by the RM

**Diversity** of solutions from different models seems to help learning over SFT

Setting	7B	7B-2	13B	13B-2	33B
Pretrain loss	1.8	1.75	1.73	1.68	1.62
ICL	11.0/18.1	14.6/-	17.8/29.3	28.7/-	35.6/53.1
SFT	35.9/48.7	41.6/55.4	43.0/55.2	50.0/61.7	<b>54.6/-</b>
RFT $k = 100$	<b>41.7/52.7</b>	<b>47.5/58.7</b>	<b>49.1/59.9</b>	<b>54.8/65.4</b>	54.5/-
Correct paths per question	53.3	60.8	62.5	71.6	88.7
Distinct paths per question	5.25	5.19	5.26	5.29	2.78

Table 1: The performance of RFT with  $k = 100$  on GSM8K compared with SFT and ICL. Distinct path amount means distinct equation list amount here.

# Next time

---

## Automated training of PRM

### **Improve Mathematical Reasoning in Language Models by Automated Process Supervision**

Liangchen Luo<sup>1\*</sup>, Yinxiao Liu<sup>1\*</sup>, Rosanne Liu<sup>1</sup>, Samrat Phatale<sup>1</sup>, Harsh Lara<sup>1</sup>, Yunxuan Li<sup>2</sup>, Lei Shu<sup>1</sup>, Yun Zhu<sup>1</sup>, Lei Meng<sup>2</sup>, Jiao Sun<sup>2</sup> and Abhinav Rastogi<sup>1</sup>