CS 957, System-2 AI
Test-Time Scaling

Mahdieh Soleymani

April 2025

Sharif University of Technology

Reasoning Models

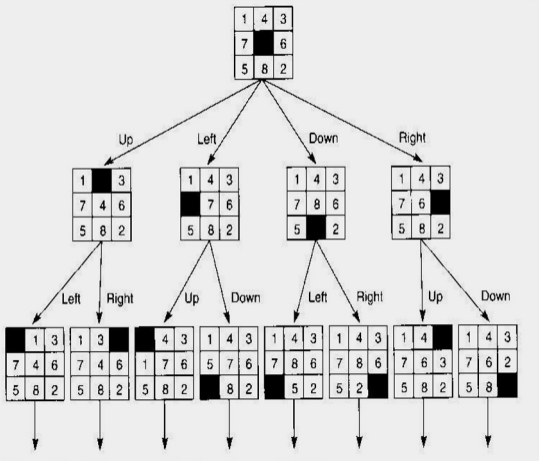
- Reasoning models solve multi-step problems by generating intermediate steps
- Examples: complex tasks such as puzzles, coding challenges, and mathematical problems.

Reasoning (and planning) vs. Learning

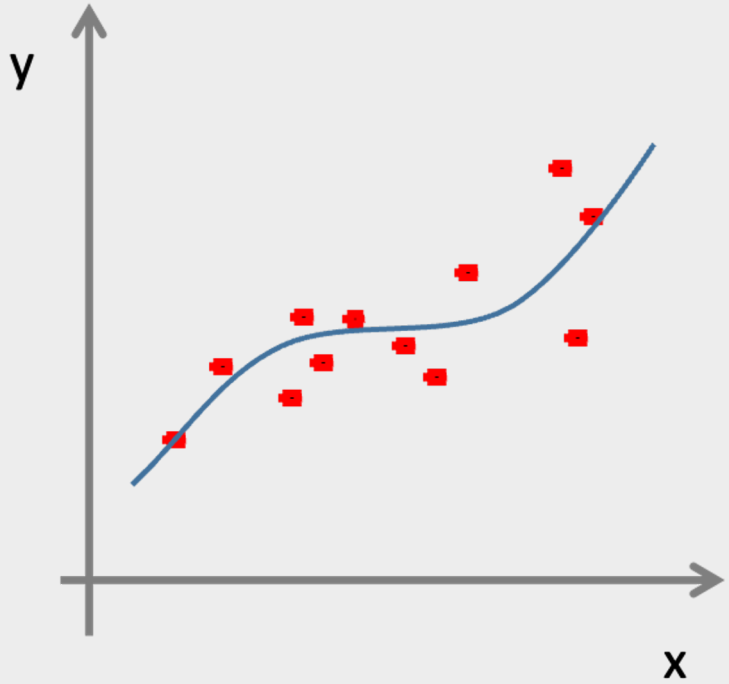
Multi-step **reasoning** to discover new facts from the existing ones

$P \wedge Q \Longrightarrow R$
$S \wedge T \Longrightarrow Q$
P
S
T

$\Big\} \; R$



Learning generalizable associations from data



The more dominant paradigms from 1950 to 2025

Symbolic AI (1950s-1980s)

Connectionism (1980s - Early 2000s)

Statistical **Machine Learning** (1990s - 2010s)

Deep Learning (2010s-present)

LLMs as a bridge between symbolic AI and DL (2020s-2025)

Symbolism vs. Connectionism

Symbolic

knowledge and its use in **reasoning** and learning (with only modest input data)

Reasoning

Search or planning

System-2

Connectionist

learning associations from data as the basis of intelligence (with little or no prior knowledge)

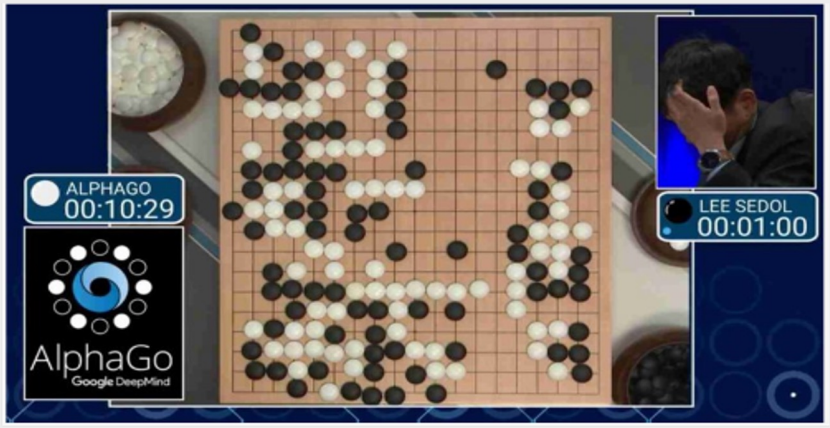
Perception ?

Learning

System-1

DL + Reinforcement Learning (RL): Evolution in Multi-step Problem Solving

- Representation learning by **DL** (to also map symbolic data to a **semantic space**)
- Reinforcement Learning (**RL**) for multi-step decision-making problems
- Learning to solve search problems more efficiently
  - reduce the cost of solving these problems independently by search

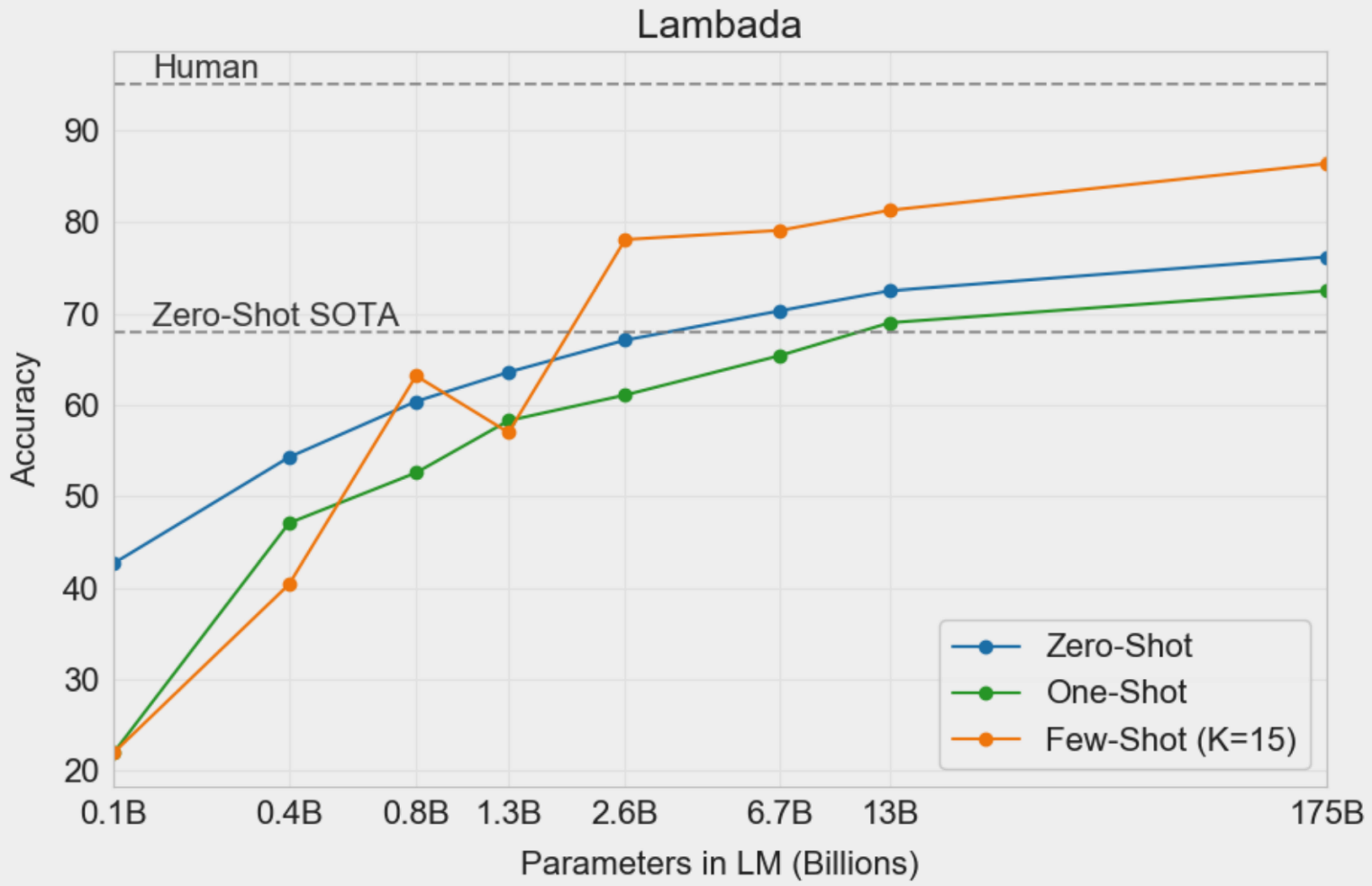


OpenAI GPT (Generative Pretraining Transformer)

- GPT (2018): training a language model **117M** on **4GB** data
  - Improving Language Understanding by Generative Pre-Training

- GPT-2 (2019): a larger version **1.5B** on **40GB** data
  - Language Models are Unsupervised Multi-task Learners

- GPT3 (2020): a more larger version **175B** on over **600GB**
  - Language Models are Few-Shot Learners



T. B. Brown et al., Language Models are Few-Shot Learners, 2023

OpenAI GPT (Generative Pretraining Transformer)

- GPT (2018): training a language model **117M** on **4GB** data
- GPT-2 (2019): a larger version **1.5B** on **40GB** data
- GPT3 (2020): a more larger version **175B** on over **600GB**
- InstructGPT, Aligning language models to follow instructions
- ChatGPT, RLHF, Business Idea, More Data and Fund
- More improvements in newer versions

Instruction

Instruction	Input	Output
Translate English into Farsi	*LLMs have revolutionized AI by enabling machines to understand and generate text*	مدل‌های زبان بزرگ (LLM) با توانمندسازی ماشین‌ها برای درک و تولید متن، انقلاب بزرگی در هوش مصنوعی ایجاد کرده‌اند.
Summarize in just 10 words to make the message even more brief.	*Welcome to the Large Language Models course at Sharif University of Technology! This course delves into the fascinating world of LLMs, a pivotal area of artificial intelligence that has transformed the field of natural language processing (NLP). Over the semester, we will explore the foundations and practical applications of these cutting-edge models.*	*Sharif University's LLM course: explore foundations and applications of LLMs*

Transfer to new tasks

Instruction	Input	Output
Summarize in Farsi to make the message more brief.	*Welcome to the Large Language Models course at Sharif University of Technology! This course delves into the fascinating world of LLMs, a pivotal area of artificial intelligence that has transformed the field of natural language processing (NLP). Over the semester, we will explore the foundations and practical applications of these cutting-edge models.*	درس LLMs دانشگاه شریف : کاوش پایه‌ها و کاربردهایLLMs

generalizes to new tasks in a zero-shot manner

Language Distinguished Properties + DL Power

- Representation learning by DL to also map symbolic data to a semantic space
  - Utilizing generalization power of DL in symbolic world too

- Distinguished properties of linguistic description
  - **Compositionality** of language
  - **Task description by language** which makes explicit relations of tasks

Enabling accurate predictions **without explicit training on specific tasks**
(as opposed to Narrow task systems)

AI Generalist Approach

LLMs for Reasoning Tasks

- LLMs can make a bridge of symbolism and connectionism
- Limited reasoning and procedural abilities
  - It still makes arithmetic errors that a calculator would avoid
  - reasoning performance drops significantly when handling out-of-distribution datasets

The Bitter Lesson

The bitter lesson is based on the historical observations that 1) AI researchers have often tried to build knowledge into their agents, 2) this always helps in the short term, and is personally satisfying to the researcher, but 3) in the long run it plateaus and even inhibits further progress, and 4) **breakthrough progress eventually arrives by an opposing approach based on scaling computation by search and learning.**

[Sutton, 2019]

General Intelligence and TTS

- General intelligence by TTS
  - transformative impact on reasoning intensive tasks
  - realizing the full promise of TTS remains a central pillar in advancing AGI
  - fully elicit the intelligence encoded in LLMs at inference time

- Notably, some studies observe patterns akin to scaling laws
  - increasing test-time compute yields consistent performance improvements.

Zhang et al., What, How, Where, and How Well? A Survey on Test-Time Scaling in Large Language Models, 2025

Test-Time Scaling (TTS)

- Importance of TTC for game playing was discovered 2016
- enhances the adaptability of LLMs by dynamically adjusting computation during inference

Spend more $ to get higher accuracy

Training $ is moving to inference $

Kumar et al., LLM Post-Training: A Deep Dive into Reasoning Large Language Models, 2025

Reasoning Intensive Tasks

- Mathematical Reasoning
- Programming & Code Generation
- Game Playing and Strategic Reasoning
- Scientific Reasoning

AIME

For any finite set X, let $|X|$ denote the number of elements in X. Define

$$S_n = \sum |A \cap B|,$$

where the sum is taken over all ordered pairs (A, B) such that A and B are subsets of $\{1, 2, 3, \cdots, n\}$ with $|A| = |B|$. For example, $S_2 = 4$ because the sum is taken over the pairs of subsets

$$(A, B) \in \{(\emptyset, \emptyset), (\{1\}, \{1\}), (\{1\}, \{2\}), (\{2\}, \{1\}), (\{2\}, \{2\}), (\{1, 2\}, \{1, 2\})\}$$

giving $S_2 = 0 + 1 + 0 + 0 + 1 + 2 = 4$. Let $\frac{S_{2022}}{S_{2021}} = \frac{p}{q}$, where p and q are relatively prime positive integers. Find the remainder when $p + q$ is divided by 1000.

D. Hendrycks et al., Measuring mathematical problem solving with the MATH dataset, 2021

TTS approaches

- **Explicit** TTS
  - Sampling or search-based methods
- **Implicit** TTS
  - Training to generate longer chains
    - SFT
    - RL

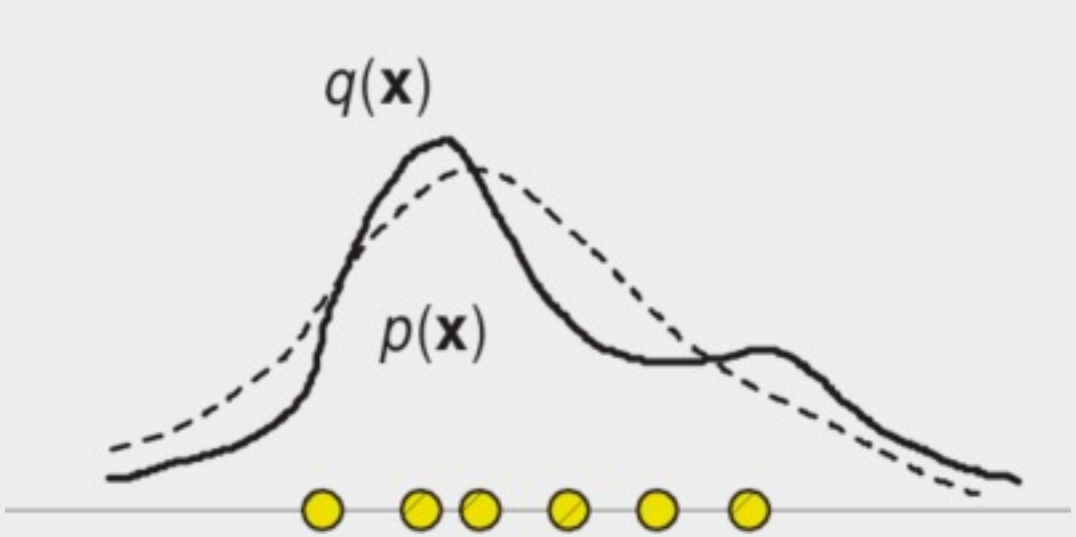
Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters, 2024

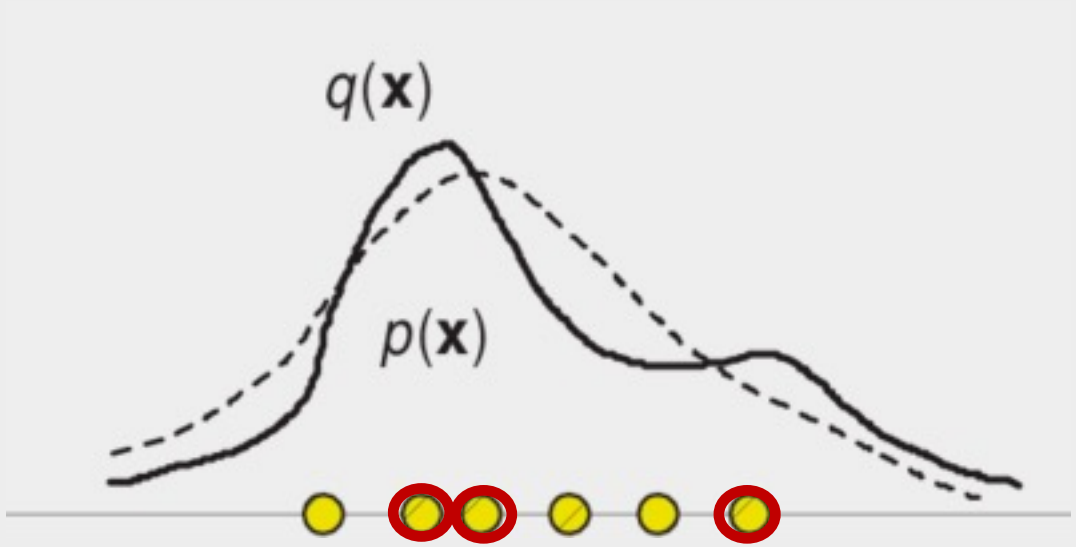
How to propose various reasoning paths?

Two axes for inducing modifications to an LLM's distribution:

- **Input domain**: Prompting (modifying proposal distribution)
  - conditioning on the prompt
- **Output domain**: Decoding or sampling
  - post-hoc verifiers or scorers to perform output modifications

LLM as a proposal distribution





How to propose various reasoning paths?

Two axes for inducing modifications to an LLM's distribution:

- Input domain: **Prompting**
- Output domain: Decoding or sampling

Reasoning (and planning) by LLMs

- LLMs + **Prompting**: Chain-of-Thought (CoT)
  - CoT simply extends their ability to do more sophisticated pattern matching.

J. Wei et al., “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models”, 2022

Recap: Chain of Thought (CoT)

(a) Few-shot

Question: 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?

Answer: The answer is **11**.

Question: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

Answer:

(Output) The answer is 8. ✘

(b) Few-shot-CoT

Question: 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?

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

Question: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

Answer:

(Output) The juggler can juggle 16 balls. Half of the balls are golf balls. So there are 16 / 2 = 8 golf balls. Half of the golf balls are blue. So there are 8 / 2 = 4 blue golf balls. The answer is 4. ✓

Step-by-step Answer

Recap: Chain of Thought (CoT)

Zero-shot

Question: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

Answer: The answer (arabic numerals) is

(Output) 8 ✘

Zero-shot-CoT

Question: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

Answer: Let's think step by step.

(Output) There are 16 balls in total. Half of the balls are golf balls. That means that there are 8 golf balls. Half of the golf balls are blue. That means that there are 4 blue golf balls. ✓

Kojima et al.,"Large Language Models are Zero-Shot Reasoners", NeurIPS 2022

Recap: Chain of Thought (CoT)

A chain of thought is **a series of intermediate natural language reasoning steps** that lead to the final output.

⟨**input**, **output**⟩ demonstrations are replaced with ⟨**input**, **chain of thought**, **output**⟩

Benefits:

- Decomposition -> easier intermediate problems
- Interpretable
- Leveraging prompting of LLM

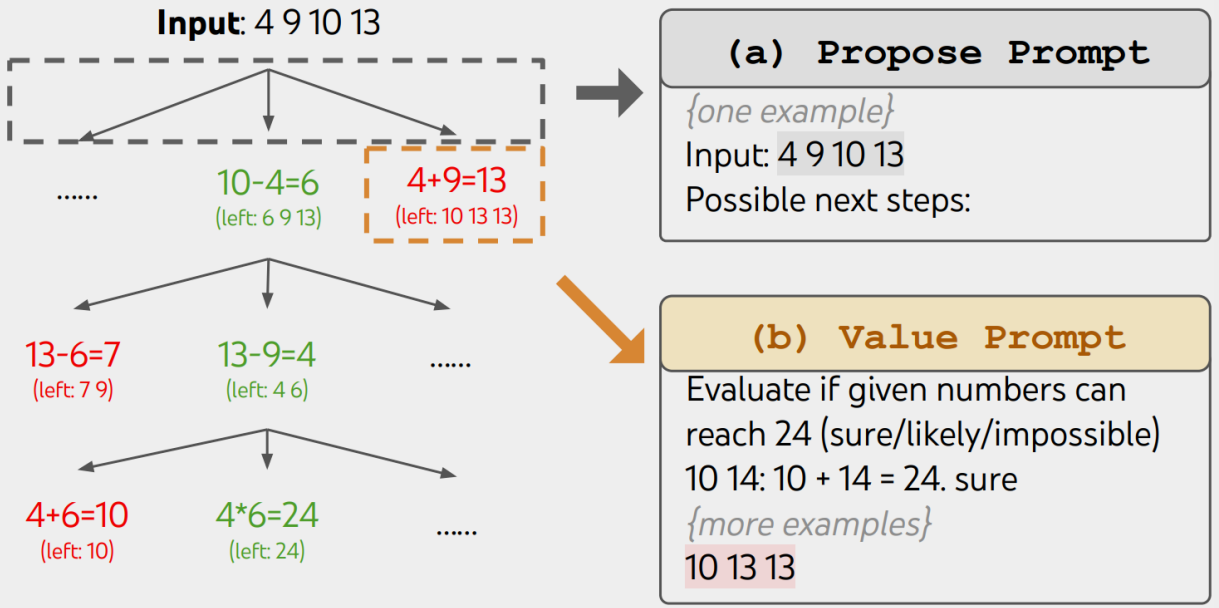
Compositionality of Language

- Compositionality of the languages
- Problem decomposition can help
  - Decompose multi-step reasoning into intermediate steps

Recap: Reasoning (and planning) by LLMs

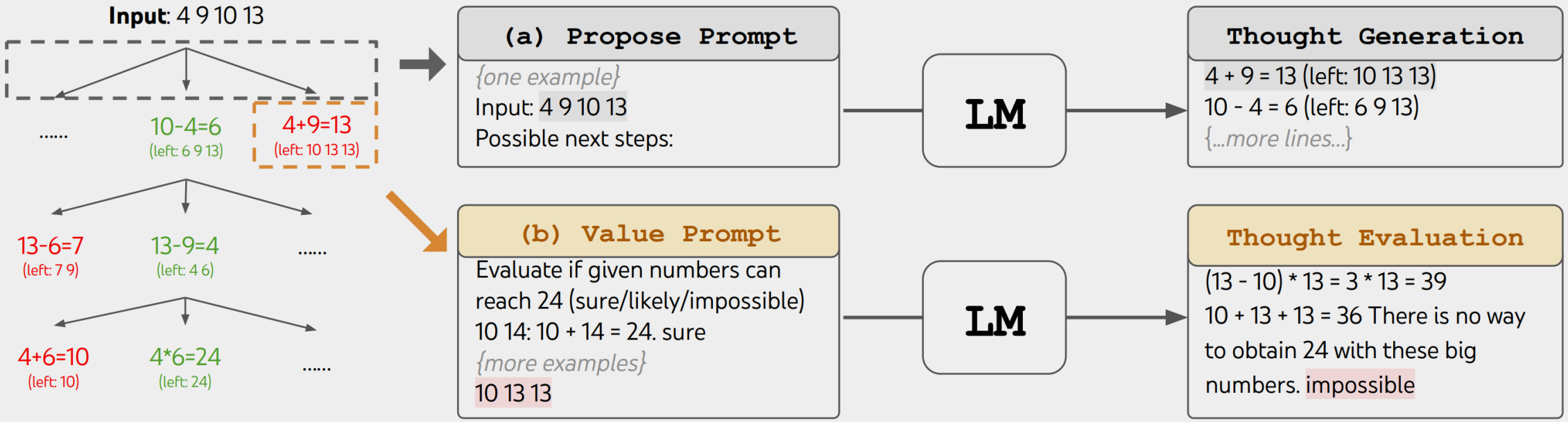
- LLMs + Prompting: Chain-of-Thought (CoT)
  - CoT simply extends their ability to do more sophisticated pattern matching.

- **LLMs + Prompting + Search**: Test-time computation by search methods
  - LLMs provide heuristics to search



S. Yao, et al Tree of Thoughts: Deliberate Problem Solving with Large Language Models, 2023.

Recap: Tree-of-Thoughts (ToT)



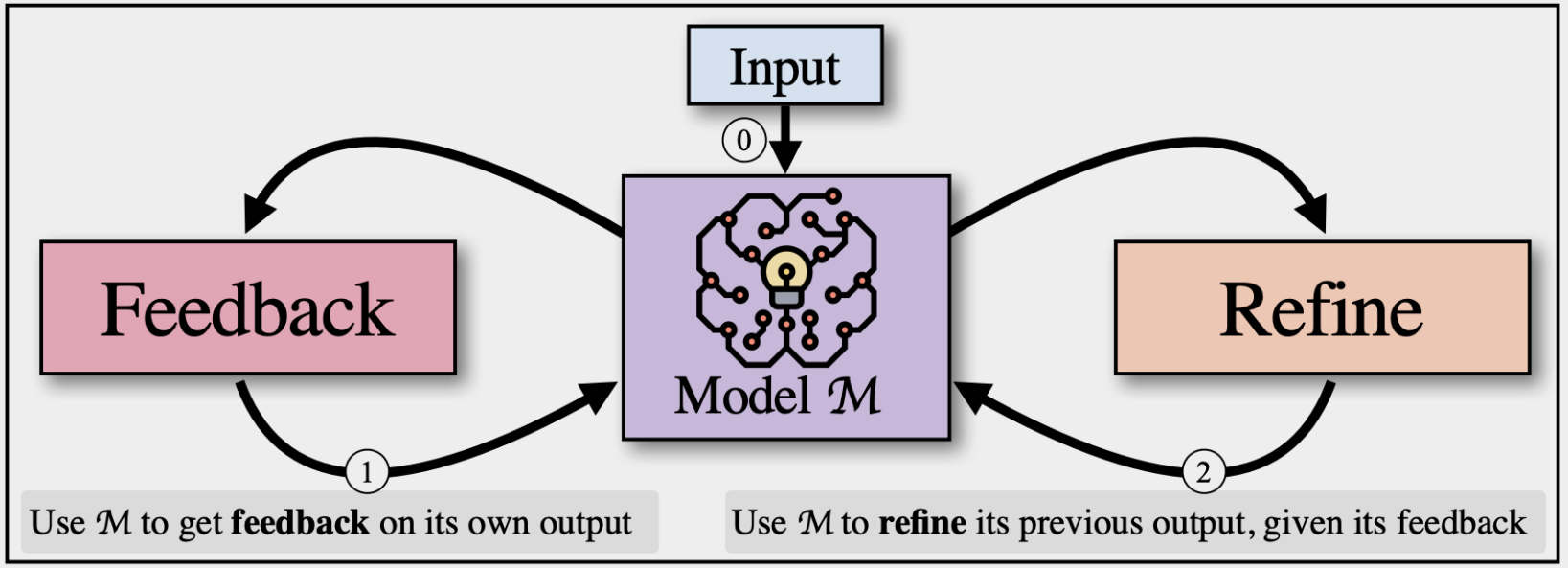
S. Yao, et al Tree of Thoughts: Deliberate Problem Solving with Large Language Models, 2023.

Self-Improvement via Refinements

- By modifying the proposal distribution from which responses are obtained by conditioning
  - for instance, by asking the base model to revise its original responses “sequentially”

Self-Refine

- LLM as the generator, refiner and the feedback provider
- Few-shot prompting to guide M to both generate feedback and incorporate the feedback into an improved draft



Madaan et al., SELF-REFINE: Iterative Refinement with Self-Feedback, 2023

Self-Refine

1) **Initial Generation**: The model produces a reasoning path.
2) **Self-Critique**: The model reviews its own response and identifies errors, inconsistencies, or areas for improvement.
3) **Refinement**: The model adjusts its response based on the critique and generates an improved version.
4) **Iteration**: The process repeats until the output meets a predefined quality threshold or stops improving.

Madaan et al., SELF-REFINE: Iterative Refinement with Self-Feedback, 2023

Algorithm 1 SELF-REFINE algorithm

Require: input x, model $\mathcal{M}$, prompts $\{p_{\text{gen}}, p_{\text{fb}}, p_{\text{refine}}\}$, stop condition $\text{stop}(\cdot)$

$y_0 = \mathcal{M}(p_{\text{gen}} \| x)$ ▷ Initial generation (Eqn. 1)

for iteration t $\in 0, 1, \ldots$ **do**

  $fb_t = \mathcal{M}\left(p_{\text{fb}} \| x \| y_t\right)$ ▷ Feedback (Eqn. 2)

  if $\text{stop}(fb_t, t)$ **then** ▷ Stop condition

    break

  else

    $y_{t+1} = \mathcal{M}\left(p_{\text{refine}} \| x \| y_0 \| fb_0 \| \ldots \| y_t \| fb_t\right)$ ▷ Refine (Eqn. 4)

  end if

end for

return y_t

Madaan et al., SELF-REFINE: Iterative Refinement with Self-Feedback, 2023

(a) **Dialogue:** x, $\mathbf{y_t}$

```
User: I am interested
in playing Table
tennis.

Response: I'm sure
it's a great way to
socialize, stay active
```

(b) **FEEDBACK** **fb**

```
Engaging: Provides no
information about table
tennis or how to play it.

User understanding: Lacks
understanding of user's
needs and state of mind.
```

(c) **REFINE** $\mathbf{y_{t+1}}$

```
Response (refined): That's
great to hear (...) ! It's
a fun sport requiring
quick reflexes and good
hand-eye coordination.
Have you played before, or
are you looking to learn?
```

(d) **Code optimization:** x, $\mathbf{y_t}$

```
Generate sum of 1, ..., N
def sum(n):
    res = 0
    for i in range(n+1):
        res += i
    return res
```

(e) **FEEDBACK** **fb**

```
This code is slow as
it uses brute force.
A better approach is
to use the formula
... (n(n+1))/2.
```

(f) **REFINE** $\mathbf{y_{t+1}}$

```
Code (refined)

def sum_faster(n):
  return (n*(n+1))//2
```

Madaan et al., SELF-REFINE: Iterative Refinement with Self-Feedback, 2023

	GPT-3.5		ChatGPT		GPT-4	
Task	Base	+SELF-REFINE	Base	+SELF-REFINE	Base	+SELF-REFINE
Sentiment Reversal	8.8	**30.4** (↑21.6)	11.4	**43.2** (↑31.8)	3.8	**36.2** (↑32.4)
Dialogue Response	36.4	**63.6** (↑27.2)	40.1	**59.9** (↑19.8)	25.4	**74.6** (↑49.2)
Code Optimization	14.8	**23.0** (↑8.2)	23.9	**27.5** (↑3.6)	27.3	**36.0** (↑8.7)
Code Readability	37.4	**51.3** (↑13.9)	27.7	**63.1** (↑35.4)	27.4	**56.2** (↑28.8)
Math Reasoning	**64.1**	**64.1** (0)	74.8	**75.0** (↑0.2)	92.9	**93.1** (↑0.2)
Acronym Generation	41.6	**56.4** (↑14.8)	27.2	**37.2** (↑10.0)	30.4	**56.0** (↑25.6)
Constrained Generation	28.0	**37.0** (↑9.0)	44.0	**67.0** (↑23.0)	15.0	**45.0** (↑30.0)

Madaan et al., SELF-REFINE: Iterative Refinement with Self-Feedback, 2023

Self-Refine: Importance of actionable feedback

- Even generic feedback provides some benefit, but the best results are achieved with targeted, constructive feedback.
- For example, in the Code Optimization task:
  - Actionable feedback: “Avoid repeated calculations in the for loop”, pinpoints an issue and suggests a clear improvement.
  - Generic feedback: “Improve the efficiency of the code”, lacks this precision and direction.

Task	SELF-REFINE feedback	Generic feedback	No feedback
Code Optimization	**27.5**	26.0	24.8
Sentiment Reversal	**43.2**	31.2	0
Acronym Generation	**56.4**	54.0	48.0

Madaan et al., SELF-REFINE: Iterative Refinement with Self-Feedback, 2023

Self-Refine: Iteration-wise score improvements

Task	y_0	y_1	y_2	y_3
Code Opt.	22.0	27.0	27.9	**28.8**
Sentiment Rev.	33.9	34.9	36.1	**36.8**
Constrained Gen.	29.0	40.3	46.7	**49.7**

Madaan et al., SELF-REFINE: Iterative Refinement with Self-Feedback, 2023

Self-Refine: Further Analysis

- Does SELF-REFINE work with weaker models?

- When SELF-REFINE failed to improve the original generation, **the majority of issues were due to erroneous feedback** rather than faulty refinements.
  - 33% of unsuccessful cases were due to feedback inaccurately pinpointing the error's location
  - 61% were a result of feedback suggesting an inappropriate fix.
  - Only 6% of failures were due to the refiner incorrectly implementing good feedback.

- These observations highlight the vital role of accurate feedback plays in SELF-REFINE.

How to improve reasoning paths?

- Prompting
  - CoT: prompts can guide the model toward step-by-step reasoning
  - Refine solutions: prompt sequentially, in order to mimic refinement process
  - Break down problems systematically: Restructuring problems before solving them to improve Models comprehension
- Decoding or sampling

How to propose various reasoning paths?

- Prompting
- **Decoding or sampling**

Decoding Recap: What is a language model?

- A probabilistic model that assigns a probability $P(w_1, w_2, \ldots, w_n)$ to every finite sequences of tokens

- Generation from a language model: $x_{1:L} \sim p$

Decoding Recap: Autoregressive language models

$$P(x_1, \dots, x_T) = P(x_1) \prod_{t=2}^{T} P(x_t | x_1, \dots, x_{t-1})$$

- $P(x_t | x_{1:t-1})$ is modeled efficiently (e.g., using a feedforward NN)

- Example:

$$P(He\ wants\ to\ know\ it)$$

$$= P(He)P(wants|He)P(to|He\ wants)P(know|He\ wants\ to)P(it|He\ wants\ to\ know)$$

Decoding Recap: Autoregressive Decoding

for $i = 1, \ldots, L$:

$$x_i \sim p_T(.|x_{1:i-1})$$

- $p_T(x_i|x_{1:i-1}) \propto p(.|x_{1:i-1})^{1/T}$ is an annealed conditional probability distribution
  - $T \geq 0$ controls randomness

$$p_T(x_i = k|x_{1:i-1}) = \frac{\exp(a_k/T)}{\sum_{j=1}^{M} \exp(a_j/T)}$$

- $T = 0$ corresponds to a greedy decoding
  - recursively selecting the highest probability token from the next-token distribution

Decoding Recap: MAP Decoding

$$\boldsymbol{y}^* = \underset{\boldsymbol{y}}{\operatorname{argmax}}\, p(\boldsymbol{y}|\boldsymbol{x})$$

- Approximation:
  - Greedy decoding

$$\hat{y}_t = \underset{y_t \in \mathcal{V}}{\operatorname{argmax}}\, p(y_t|\boldsymbol{x}, \hat{y}_{1:t-1})$$

  - Beam Search (We will see it later)

CoT

- x: problem specification
- $z_{1:T}$: chain of thought (CoT) steps

$$z_t \sim p(.|x, z_{1:t-1}), \quad t = 1, \ldots, T$$

T is the amount of test-time compute

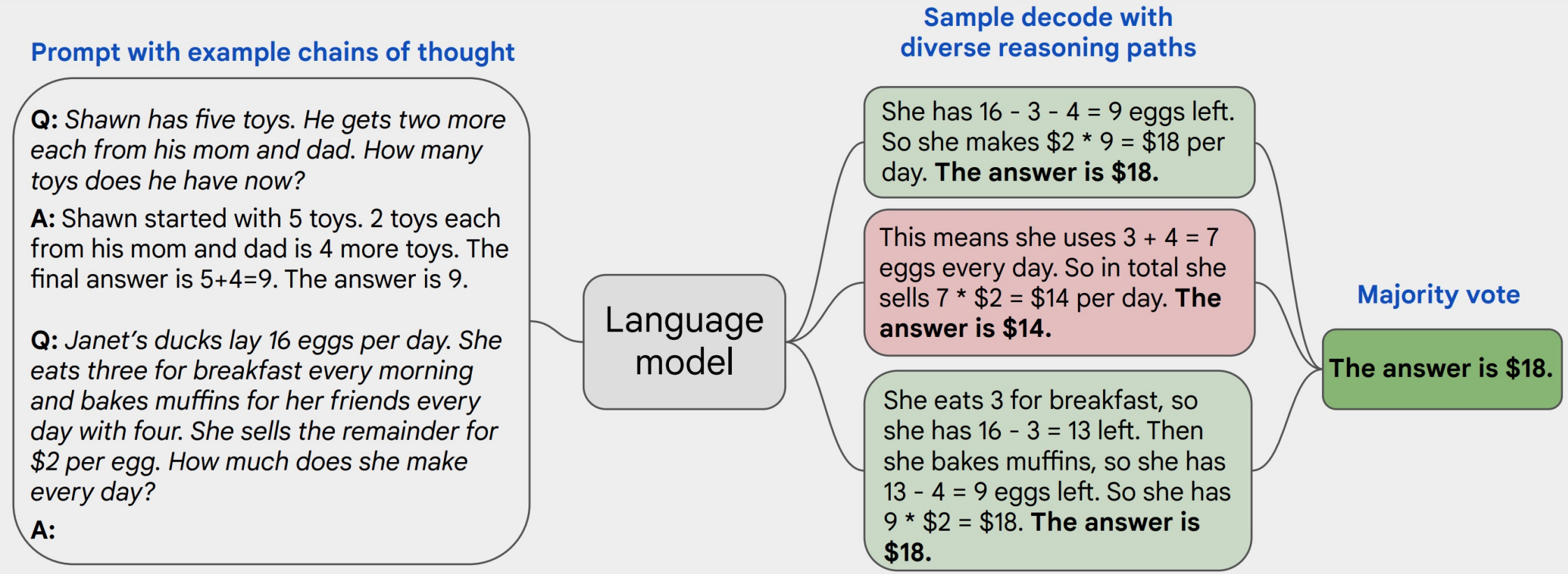
- $y \in \mathcal{Y}$: final answer

$$y \sim p(.|x, z_{1:T})$$

- $p(y|x) = \mathbb{E}_z[p(y|z_{1:T}, x)]$

Self-Consistency

Change greedy decode (single-path) to self-consistency (multi-path) in few-shot CoT:



X. Wang et al., Self-consistency improves chain of thought reasoning in language models, 2022

Self-Consistency

- Think in many ways and trust the consensus
  - turns the one-shot CoT into an ensemble and cross-verifies the answers
  - if different reasoning paths converge to the same conclusion, the answer is more likely to be correct.
  - effectively reduces variance in model outputs and mitigates occasional hallucinations

- Aggregation: majority vote or highest probability answer after marginalizing out the latent reasoning

X. Wang et al., Self-consistency improves chain of thought reasoning in language models, 2022

Self-Consistency / Majority Vote

- For $n = 1, \ldots, N$ (sampling with temperature > 0):

$$z_{1:T}^{n} \sim p(. \mid x)$$

$$y^{n} \sim p(. \mid x, z_{1:T}^{n})$$

- Pick majority among $\{y^{1}, \ldots, y^{N}\}$

Self-Consistency

Showcase results on **AR**, **CR** tasks:

	Method	GSM8K	CommonsenseQA
	Previous SoTA	35^{e} / 57^{g}	**91.2**^{a}
LaMDA (137B)	Greedy decode (Single-path)	17.1	57.9
	Self-Consistency (Multi-path)	27.7 (+10.6)	63.1 (+5.2)
PaLM (540B)	Greedy decode (Single-path)	56.5	79.0
	Self-Consistency (Multi-path)	**74.4** (+17.9)	80.7 (+1.7)

X. Wang et al., Self-consistency improves chain of thought reasoning in language models, 2022

Beam Search

- Top K partial sequences (the ‘beam’) are kept at each step

- Maintain the K most promising states at each reasoning step.
  - Pruning less promising reasoning branches

Beam Search

- For each time step:
  - Sample several times the next step of each selected path $z^k_{1:t_k}$ among the K kept ones (where t_k shows the current length of k-th path)

$$z^{k,i}_{t_k+1} \sim p\left(. \middle| x, z^k_{1:t_k}\right)$$

  - Keep the top samples among $\left\{z^{k,i}_{1:t_k+1}\right\}_{i=1,\dots,N,k=1,\dots,K}$ according a scoring function g

Beam Search

- Prioritizes (partial) reasoning paths zs based on the model's own confidence

$$g(z|x) = \log p(z|x) = \sum_{t=1}^{T} \log p(z_t|z_{1:t-1}, x)$$

Beam Search

1. Sample N initial predictions for the first step in the solution
2. Score the generated steps according to the PRM's score
3. Filter for only the top N/M highest scoring steps
4. From each candidate, sample M proposals from the next step, resulting in a total of $N/M \times M$ candidate prefixes again.
5. Repeat steps 2-4 until the maximum depth is reached or an <EOS> token is generated

Model Confidence vs. Verifier

- Model Confidence: Model's probability estimates as a heuristic guiding the search

- Prioritizes reasoning paths based on the model's own confidence
  - log probability (i.e., the model's most confident output)
  - When exploring a reasoning tree or multi-step solution, use the model's probabilities to decide which branch to expand

- External Verifier can evaluate reasoning paths instead of relying only on the LLMs's policy or distribution

Effectiveness of repeated sampling

- Assume we have automatic verification tools
  - Existing tools like proof checkers and unit tests for software

- Coverage: As the number of samples increases, what fraction of problems can we solve using any sample that was generated?

B. Brown et al., Large Language Monkeys: Scaling Inference Compute with Repeated Sampling, 2024

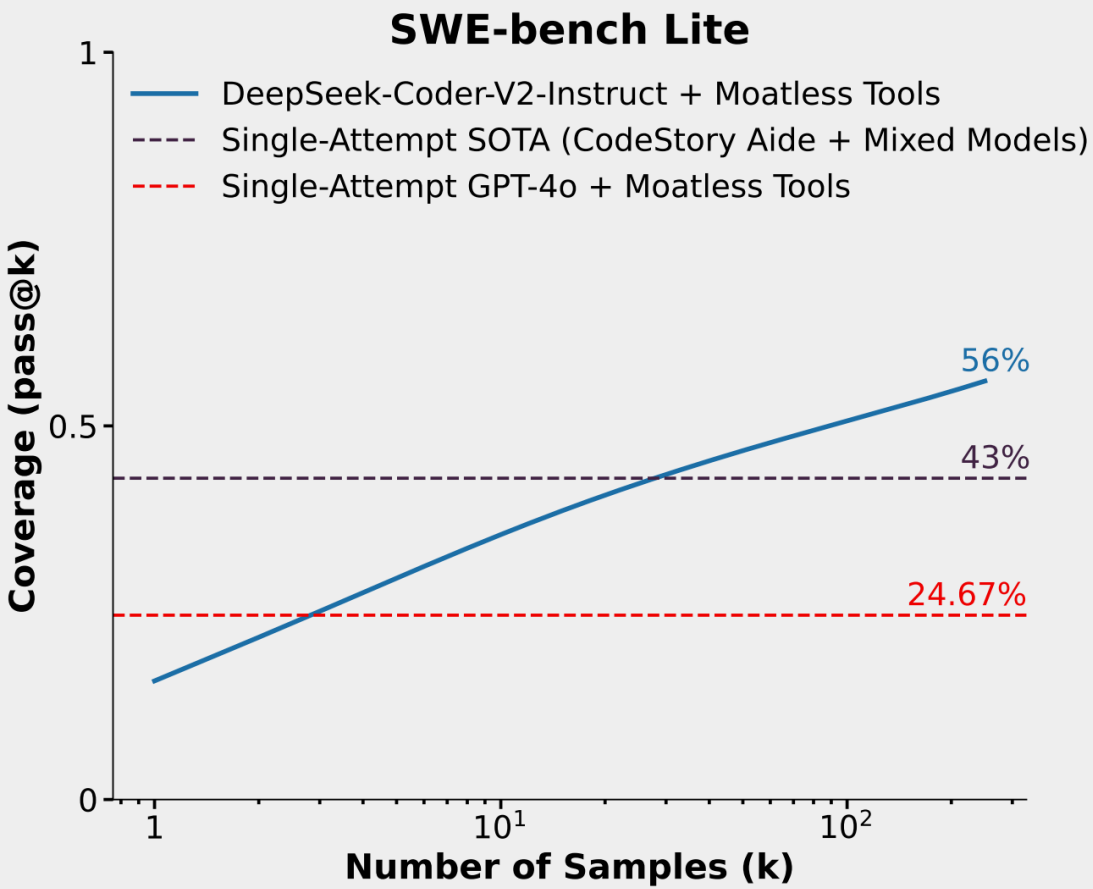
Coverage

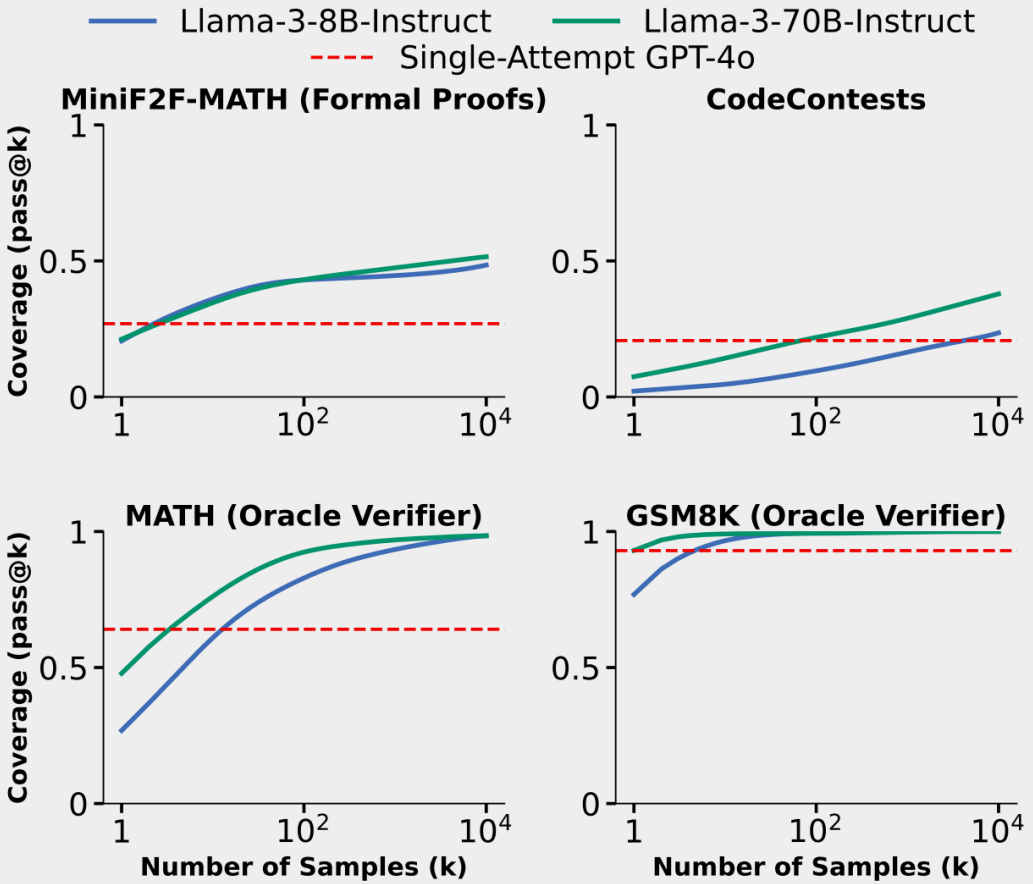
- The total fraction of problems solved among k samples for those problems
- To reduce the variance, this unbaised estimator is used:

$$\text{pass@k} = \frac{1}{\#\text{ of problems}} \sum_{i=1}^{\#\text{ of problems}} \left(1 - \frac{\binom{N-C_i}{k}}{\binom{N}{k}}\right)$$

- k denotes the number of samples per problem
- $N \geq k$ samples is generated for each problem
- C_i: the number of correct samples for problem i among N ones

B. Brown et al., Large Language Monkeys: Scaling Inference Compute with Repeated Sampling, 2024





B. Brown et al., Large Language Monkeys: Scaling Inference Compute with Repeated Sampling, 2024

Best-of-N

Sample N candidate solutions

These candidates are scored using external reward models or verifiers.

The highest-scoring solution is selected as the final answer

Verifier & Proposer

- Proposer and a Verifier to systematically explore and evaluate answers.
  - Proposer (usually the base LLM) generates multiple candidate solutions
    - often using methods like high temperature sampling or diverse decoding.
  - Verifier (another model or a heuristic) judges and selects the best.

Snell et al., Scaling llm test-time compute optimally can be more effective than scaling model parameters, 2024.

Reward Models or Verifiers

A reward function that maps response text to a scalar value

The reward model can be trained to replicate the label-based scores

Reward Models

process-based rewards (e.g., chain-of-thought reasoning) with outcome-based evaluations (e.g., response quality)

- **Outcome**-based Reward Models (**ORMs**)

$$\mathcal{L}_{ORM} = -(y_s \log r_s + (1 - y_s) \log(1 - r_s)),$$

- **Process**-based Reward Models (**PRMs**)

$$\mathcal{L}_{PRM} = -\sum_{i=1}^{K} y_{s_i} \log r_{s_i} + (1 - y_{s_i}) \log(1 - r_{s_i})$$

P. Wang et al., Math-Shepherd: Verify and Reinforce LLMs Step-by-step without Human Annotations, 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

H. Lightman et al., Let's Verify Step by Step, 2023

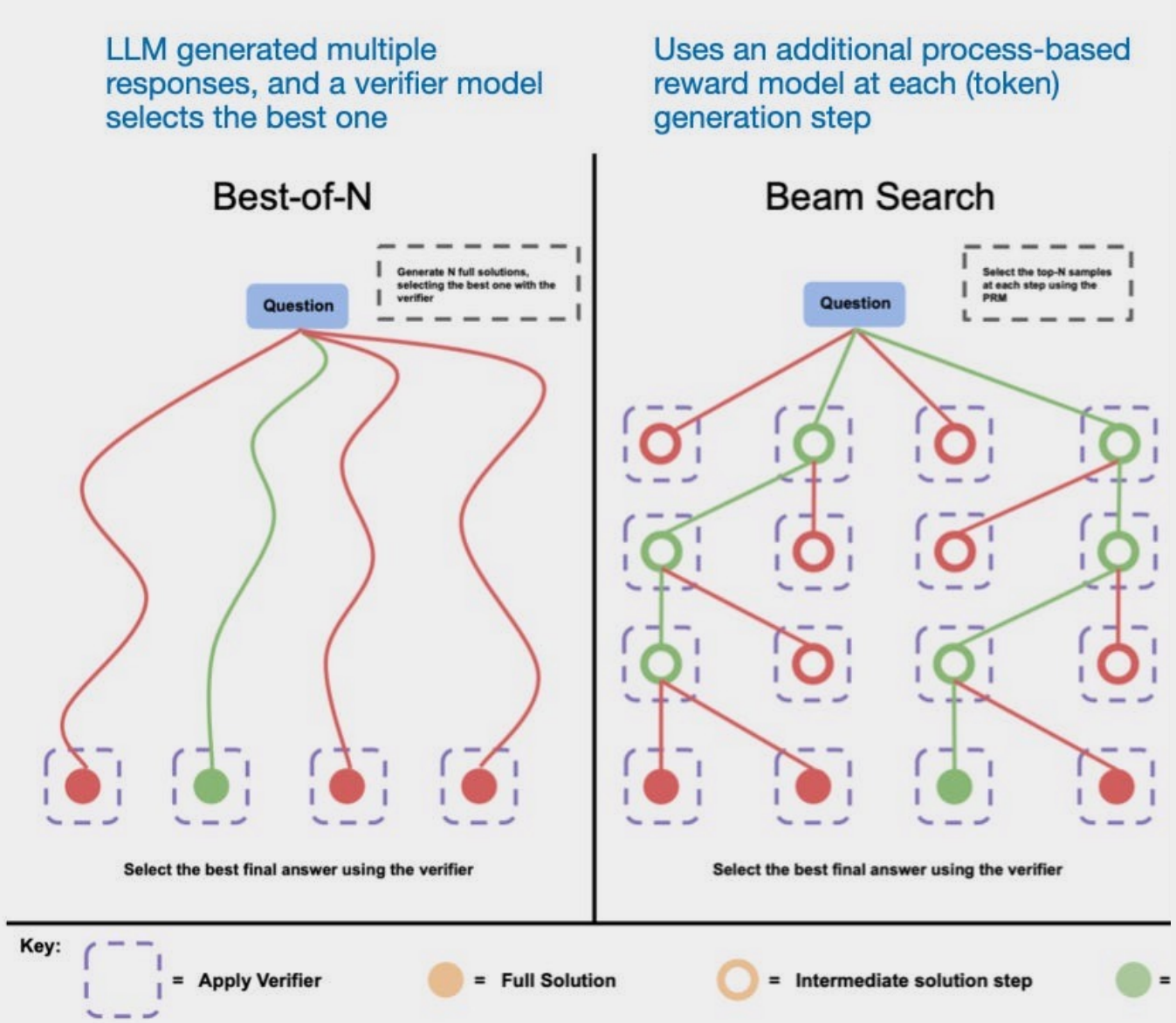
Beam Search with Guide

- $g(z|x)$ is specified by a PRM
- Prioritizes the (partial) reasoning paths zs according to a PRM
- Beam Search incrementally expands and prunes partial hypotheses

C. Snell et al., Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters, 2024

Proposer & Verifier

- A unified framework

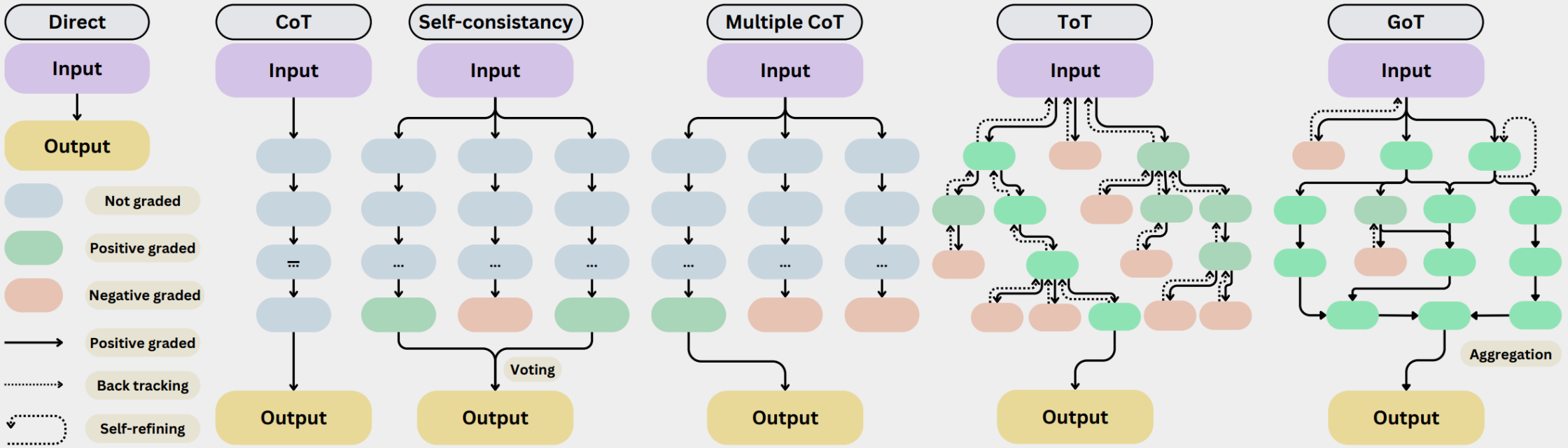


Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters, 2024

Best-of-N vs. Beam search

- Generates N candidate outputs (usually via sampling) independently and then picks the best one
  - allowing for greater diversity but at a higher computational cost.

- Beam Search systematically aims for the highest score (e.g., most probable) sequence



Kumar et al., LLM Post-Training: A Deep Dive into Reasoning Large Language Models, 2025

Aggregation & Selection

- Step-wise aggregation
  - **PRM-Minimum**: aggregating the per-step scores by taking the minimum
  - **PRM-Product**: aggregating the per-step scores by taking the product
  - **PRM-Avg**: aggregating the per-step scores by taking the average
- Inter-answer aggregation
  - Majority Vote
  - standard best-of-N
  - “best-of-N weighted”: marginalizes the verifier’s scores across all solutions with the same final answer and selects the one with the greatest total sum.