

IERG5050 AI Foundation Models, Systems & Applications Spring 2025

Teaching LLMs to Reason and Reasoning LLMs

Prof. Wing C. Lau

wclau@ie.cuhk.edu.hk

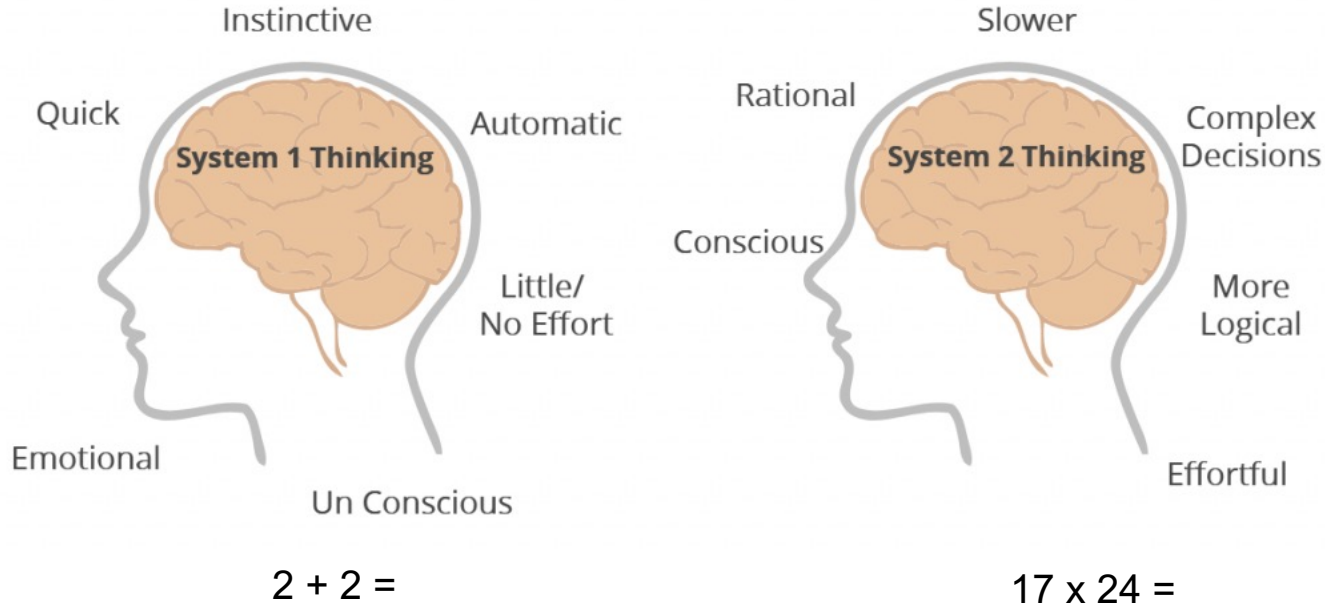
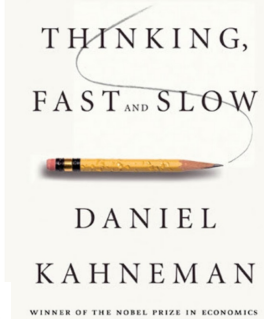
<http://www.ie.cuhk.edu.hk/wclau>

Acknowledgements

Many of the slides in this lecture are adapted from the sources below. Copyrights belong to the original authors.

- Xinyun Chen, “Inference-Time Techniques for LLM Reasoning,” Guest Lecture for UC Berkeley MOOC on Advanced Large Language Model Agents, Spring 2025, <https://www.youtube.com/live/g0Dwtf3BH-0>, https://rdi.berkeley.edu/llm-agents-mooc/slides/inference_time_techniques_lecture_sp25.pdf.
- Denny Zhou, “Teaching Language Models to Reason”, Guest Lecture for UC Berkeley MOOC on Large Language Model Agents, Fall 2024, <https://rdi.berkeley.edu/llm-agents-mooc/slides/llm-reasoning.pdf>, https://www.youtube.com/live/QL-FS_ZcmYo, <https://dennyzhou.github.io/LLM-Reasoning-Berkeley.pdf>, <https://dennyzhou.github.io/LLMs-Reason-Taiwan-2023.pdf>, <https://dennyzhou.github.io/LLMs-Reason-2023-Harvard-Yale.pdf>
- Sebastian Raschka, “Understanding Reasoning LLMs,” Ahead of AI, Feb 5, 2025, <https://magazine.sebastianraschka.com/p/understanding-reasoning-llms>
- Sebastian Raschka, “The State of LLM Reasoning Models: Inference-Time Compute Scaling Methods,” Ahead of AI, Mar 8, 2025, <https://magazine.sebastianraschka.com/p/state-of-llm-reasoning-and-inference-scaling>
- Sasha Rush, Daniel Ritter, “Speculations on Test-Time Scaling (o1),” <https://www.youtube.com/watch?v=6PEJ96k1kiw>, <https://srush.github.io/awesome-o1/o1-tutorial.pdf>, <https://github.com/srush/awesome-o1>
- Charlie Snell, “Scaling LLM Test-Time Compute,” AI4All, <https://www.youtube.com/watch?v=OXwGp9YeuBg>, <https://docs.google.com/presentation/d/1UdmRaXwcRT341G77ycnbuhyKCMFWKJ5Sxv7la2JLYEF/edit#slide=id.g>
- Charlie Snell et al, “Scaling LLM Test-Time Compute Optimally can be more effective than Scaling Model Parameters,” <https://arxiv.org/abs/2408.03314>
- Noah Goodman, “Learning to Reason, Insights from Language Modeling,” MIT Center for Brains, Minds+ Machines (MITCBMM) seminar, <https://www.youtube.com/watch?v=YR9EztQF0R8>
- Ben Prystawski, Michael Y. Li, Noah D. Goodman, “Why think step by step ? Reasoning emerges from the Locality of Experience,” NeurIPS 2023, <https://neurips.cc/media/neurips-2023/Slides/73821.pdf>, <https://neurips.cc/virtual/2023/oral/73821>
- Maarten Grootendorst, “A Visual Guide to Reasoning LLMs – Exploring Test-Time Compute Techniques and DeepSeek-R1,” Feb 3, 2025, <https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-reasoning-llms>
- Jason Weston, “Learning to Reason with LLMs,” Guest Lecture for UC Berkeley MOOC on Advanced Large Language Model Agents, Spring 2025, https://www.youtube.com/live/_MNll_hU33fH0, <https://rdi.berkeley.edu/adv-llm-agents/slides/Jason-Weston-Reasoning-Alignment-Berkeley-Talk.pdf>
- Logesh Kumar Umapathi, “Unlocking Reasoning and Planning abilities in Large Language Models,” Conf42 Machine Learning – Online, May, 2023, <https://www.youtube.com/watch?v=1K6M7o7FTy4>, https://conf42.github.io/static/slides/l_logesh%20Kumar%20Umapathi%20-%20Conf42%20Machine%20Learning%202023.pdf
- Overview of Stanford CS25:Transformer United V4, Spring 2024, https://docs.google.com/presentation/d/1oXP3LXtIVisVbwTvGIaWf_aWvak9c1uNC4uhkS6olk/edit?usp=sharing
- Aske Plaat et al, “Reasoning with Large Language Models, a Survey,” July 2024, <https://arxiv.org/pdf/2407.11511>
- Jie Huang, Kevin C.C. Chang, “Towards Reasoning in Large Language Models,” ACL Findings 2023, <https://aclanthology.org/2023.findings-acl.67.mp4>, <https://aclanthology.org/2023.findings-acl.67.pdf>
- Hanna Hajishirzi, “Open Training Recipes for Reasoning in Language Models,” Guest Lecture for UC Berkeley MOOC on Advanced Large Language Model Agents, Spring 2025, <https://rdi.berkeley.edu/llm-agents-mooc/slides/OLMo-Tulu-Reasoning-Hanna.pdf>, <https://www.youtube.com/live/cMiu3A7YBks>
- Misguided Attention: a suite of testing prompts: <https://github.com/cpldcpu/MisguidedAttention>, <https://huggingface.co/posts/Severian/375067343900874>

Two Modes of Human Thought: Thinking, FAST and SLOW System 1 vs. System 2



Early LLMs seem to only have a System 1

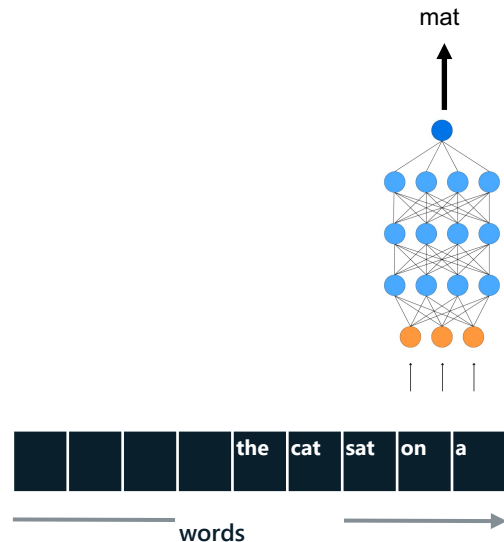
(as they were developed mainly based on the so-called “Deep Learning Hypothesis”)

Source: Ilya Sutskever: Sequence to sequence learning with neural networks, NeurIPS 2024 Test of Time Award Talk.

“The Deep Learning Hypothesis”

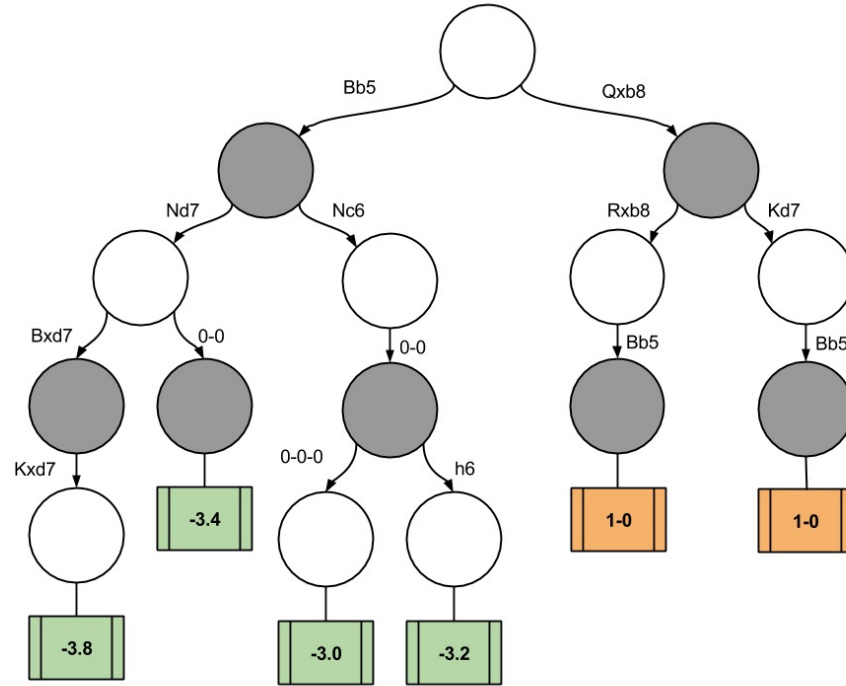
- Human perception is fast
 - Neurons fire at most 100 times a second
 - Humans solve perception in 0.1 seconds
 - our neurons fire 10 times, at most

Anything a human can do in 0.1 seconds, a big 10-layer neural network can do, too!



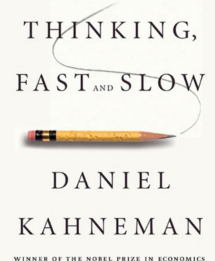
Prompting and Fine-Tuning can (and only) yield a (better) System 1 !

System 2



System 1: generates the proposals (used in speed chess)

System 2: Need to “Think” more / Reason, e.g. keep track of the tree (used in competitions)



What do we mean by an LLM knows how to “Reason” ?

“Reasoning is decomposing a potentially complex task into simpler subtasks the LM can solve more easily by itself or using tools. There exist various ways to decompose into subtasks, such as recursion or iteration. In that sense, reasoning is akin to planning.”

from [G. Mialon et al, Augmented Language Models: a Survey] of Meta AI.

What do we mean by an LLM knows how to “Reason” ?

- For LLMs, “Reasoning” is commonly defined as the process of answering questions that require complex, multi-step generation with intermediate steps.
 - As opposed to answer single-step Factual-recall type of questions

If a train is moving at 60 mph and travels for 3 hours, how far does it go?

The train travels 180 miles.

Plain response

To determine the distance traveled, use the formula:

Distance = Speed $\times$ Time

Given that the speed is 60 mph and the time is 3 hours:

Distance = 60 mph $\times$ 3 hours = 180 miles

So, the train travels 180 miles.

Response with intermediate reasoning steps

A regular LLM may only provide a short answer (as shown on the left), whereas reasoning models typically include intermediate steps that reveal part of the thought process. (Note that many LLMs who have not been specifically developed for reasoning tasks can also provide intermediate reasoning steps in their answers.)

Another Example of an LLM's Reasoning in Action

1) Intermediate reasoning chain that is not explicitly shown to the user

2) Intermediate reasoning steps that are shown as part of the answer

The main answer

ChatGPT o1

What is the last digit of 7^{999}

Reasoned about last digit of powers for a couple of seconds

To find the last digit of 7^{999} , note that the last digits of powers of 7 repeat in a cycle of length 4:

7^1	= 7	(last digit 7)
7^2	= 49	(last digit 9)
7^3	= 343	(last digit 3)
7^4	= 2401	(last digit 1)

Then it repeats: 7, 9, 3, 1, 7, 9, 3, 1, ...

To determine which of these four digits appears for 7^{999} , we look at $999 \bmod 4$:

$$999 \div 4 = 249 \text{ remainder } 3.$$

Since the remainder is 3, we take the third digit in the cycle (7, 9, 3, 1), which is 3.

Hence, the last digit of 7^{999} is 3.

Here, "Reasoning" is used at two different levels:

- 1) processing the input and generating via multiple intermediate steps and
- 2) providing some sort of reasoning as part of the response to the user

Why do we want LLMs to be able to “Reason” ?

- To increase Few-shot Learning Ability

“Humans (even a small child) can learn from relatively few examples because Human can “Reason.”” from Dr. Denny Zhou of Google DeepMind/

- To improve Out-of-distribution Robustness

Humans possess the ability to generalize beyond familiar scenarios.

- To provide Explainability

Humans are capable of providing explanations for their decisions or predictions, whereas machines (especially deep neural networks) are often considered “black box” within limited explainability.

Can LLMs really reason ?

There are some indications that LLMs are able to reason, including:

- ▶ High performance on various tasks requiring reasoning (Suzgun et al., 2022)
- ▶ The ability to reason step-by-step with chain-of-thought prompting (Wei et al., 2022)
- ▶ The reflection of human-like content effects on reasoning (Dasgupta et al., 2022)

Can LLMs really reason ?

But there are also observations that suggest LLMs may not be capable of reasoning:

- ▶ LLMs still struggle with tasks that require complex reasoning (Valmeekam et al., 2022; Han et al., 2022; Ruis et al., 2022)
- ▶ The performance of LLMs has been found to be sensitive to the frequency of certain terms (Razeghi et al., 2022; Jung et al., 2022)
- ▶ Language models have been found to struggle with associating relevant information that they have memorized (Huang et al., 2022)
 - ▶ *“It is not yet fully understood whether the LM is really reasoning, or simply producing a larger context that increases the likelihood of correctly predicting the missing tokens.”*
(G. Mialon et al., “Augmented Language Models: a Survey 2023).
 - ▶ Misguided Attention: a suite of testing prompts: <https://github.com/cpldcpu/MisguidedAttention>
- ▶ Some mistakes made by LLMs seems to show they do not actually know reasoning

Early LLMs (ChatGPT) had Unreliable Reasoning Even On Simple Tasks

how many r's are there in the word "strawberry"?

Probably due to **tokenization**!
Or was it due to Sept 11 ?



There are two "r's" in the word "strawberry."

9.11 and 9.9, which one is larger?

9.11 and 9.9, which one is larger?

Generated by gpt-4o's tokenizer.



9.11 is larger than 9.9.

Try it out at:

<https://tiktokenizer.vercel.app/>

Do LLMs really know how to “reason” ?

Even for Frontier LLMs, their responses to certain kind of prompts seem to indicate that they actually do NOT understand the real concept but merely perform “pattern matching”. In other words, they are not really “reasoning”, e.g.,

- LLMs can be easily distracted by Irrelevant Context/ Content
- Premise Order matters in LLM “Reasoning”, but they should not be !
- LLMs cannot Self-correct reliably without an oracle: while allowing LLMs to review self-generated responses can sometimes help to correct inaccurate answers, such review may also change correct answers into incorrect ones.

References:

- Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed Chi, Nathanael Schärli, and Denny Zhou. Large Language Models Can Be Easily Distracted by Irrelevant Context. ICML 2023.
- Xinyun Chen, Ryan A Chi, Xuezhi Wang, Denny Zhou. Premise Order Matters in Reasoning with Large Language Models. ICML 2024.
- Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, Denny Zhou. Large Language Models Cannot Self-Correct Reasoning Yet. ICLR 2024.
- Misguided Attention: a suite of testing prompts: <https://github.com/cpldcpu/MisguidedAttention>

LLM readily distracted by Irrelevant Context/ Content

Adding irrelevant contexts to GSM8K leads to 20+ points performance drop

Lucy has \$65 in the bank. She made a \$15 deposit and then followed by a \$4 withdrawal. Maria's monthly rent is \$10. What is Lucy's bank balance?

LLM

Lucy's bank balance is $\$65 + \$15 - \$4 - \$10 = \$66$. The answer is \$66.

LLM readily distracted by Irrelevant Context/ Content

- ❖ Simply adding irrelevant sentences can still hurt performance, e.g.

The sky is blue. The sky is blue. The sky is blue. The sky is blue.

Lucy has \$65 in the bank. The sky is blue. The grass is green. The grass is green. **She made a \$15 deposit and then followed by a \$4 withdrawal.** The sky is blue. The sky is blue. The sky is blue. The sky is blue. The grass is green. The grass is green. The sky is blue. The grass is green. **What is Lucy's bank balance?**

LLM readily distracted by Irrelevant Context/ Content

BUT

Prompting LLMs to Ignore irrelevant context can take some performance back

Lucy has \$65 in the bank. She made a \$15 deposit and then followed by a \$4 withdrawal.
Maria's monthly rent is \$10. What is Lucy's bank balance? Ignore irrelevant context.

LLM

Lucy has \$65 in the bank. After making a \$15 deposit and then a \$4 withdrawal, her bank balance is \$76. Maria's monthly rent is not relevant to this question, so it can be ignored. The answer is \$76.

Premise Order matters for LLMs

[Original GSM8K Problem] Thomas withdraws \$1000 in 20 dollar bills from the bank account. He loses 10 bills while getting home. After that, he uses half of the remaining bills to pay for a bill. Thomas then triples his money. He then converts all his bills to 5 dollar bills. How many 5 dollar bills does he have?

[Reordered GSM8K Problem] Thomas withdraws \$1000 in 20 dollar bills from the bank account. After getting home, he uses half of the remaining bills to pay for a bill. Thomas then triples his money. He then converts all his bills to 5 dollar bills. He loses 10 bills while getting home. How many 5 dollar bills does he have?

About 10 points drop on solving rates across all frontier LLMs

Premise Order matters for LLMs

Response for original

1. Thomas withdraws \$1000 in \$20 bills. $\$1000 / \$20 = 50$ bills
2. He loses 10 of those bills. 50 bills - 10 bills = 40 bills
3. He uses half of the remaining bills to pay for a bill. Half of 40 bills is 20 bills. 40 bills - 20 bills = 20 bills remaining
4. Thomas then triples his money. He has 20 bills of \$20, which is $20 * \$20 = \400 . Tripling \$400 gives us $\$400 * 3 = \1200 .
5. He converts all his money to \$5 bills. $\$1200 / \$5 = 240$ bills

Response for reordered

1. Thomas withdraws \$1000 in \$20 bills from the bank. $\$1000 / \$20 = 50$ bills
2. He uses half of the remaining bills to pay for a bill. Half of 50 bills is 25 bills. So, he uses 25 bills and has 25 bills left.
3. Thomas then triples his money. Since he had 25 bills of \$20 each, he had $25 * \$20 = \500 before tripling his money. Tripling \$500 gives him $\$500 * 3 = \1500 .
4. He then converts all his bills to \$5 bills. $\$1500 / \$5 = 300$ bills
5. He lost 10 bills while getting home. 300 bills - 10 bills = 290 bills

Premise Order matters for LLMs

Rules: If duve, then dax. If kere and riff, then tess. If feg and tess, then rabe. If wozz, then riff. If rabe, then thoxe. If ag and riff and kere, then hepe. If thoxe, then vepe. If nife and hume and ag, then dax. If feg and thoxe, then ire. If rebe and ene and heff, then kone. If ire and dax, then wope. If tess and vepe, then nup. If rabe and vide and nife, then ag. If nup and ag and vide, then duve. If zeck and hepe and dax, then riff. If nup, then hepe. If sene, then hume. If hepe and tess and ag, then kere. If fum, then vide. If kere and wope, then fum. If jag and kone, then thoxe. If fum, then wozz.

Facts: Alice is cote. Alice is kone. Alice is duve. Alice is ag. Alice is jag. Alice is tess. Alice is riff. Alice is feg. Alice is vide.

Query: Is Alice wozz? In each step, indicate whether you are using a fact, or a rule.

Logical inference task: The rules are ordered according to their use in the inference process, though not all rules are necessary for the query

- ❖ When the rules relevant to the query are **randomly** ordered, 30+ points performance drop across all frontier LLMs !

LLM cannot self-correct (reasoning) reliably without an Oracle

Christina is planning a birthday party and needs .75 gift bags per invited guest, because 1/4 of attendees don't show up. She invited 16 friends. Gift bags are \$2 each. How much will she spend?

... $0.75 * 12 = 9$ gift bags ... Christina will spend a total of $2 * 9 = \$8$... (wrong)

Review your previous answer and find problems with your answer.

Upon reviewing my previous answer, I recognize a mistake. I incorrectly stated that Christina needs 0.75 gift bags per invited guest. The given information states that 1/4 of the attendees won't show up...

Based on the problems you found, improve your answer.

... $16 * (3/4) = 12$ gift bags ... Christina will spend $12 * \$2 = \24 ... (Correct)

LLM cannot self-correct (reasoning) reliably without an Oracle

Terry eats 2 yogurts a day. They are currently on sale at 4 yogurts for \$5.00. How much does he spend on yogurt over 30 days?

... So, Terry spends \$75 on yogurt over 30 days. (Correct)

Review your previous answer and find problems with your answer.

Upon reviewing my previous answer, I realized that I made a mistake in calculating Terry's spending on yogurt over 30 days. I incorrectly stated that Terry spends \$2.50 per day for 30 days...

Based on the problems you found, improve your answer.

... the final answer is Terry spends \$37.5 on yogurt over 30 days. (wrong)

Self-correcting leads to Worse Results

		# calls	GSM8K	CommonSenseQA	HotpotQA
GPT-3.5	Standard Prompting	1	75.9	75.8	26.0
	Self-Correct (round 1)	3	75.1	38.1	25.0
	Self-Correct (round 2)	5	74.7	41.8	25.0
GPT-4	Standard Prompting	1	95.5	82.0	49.0
	Self-Correct (round 1)	3	91.5	79.5	49.0
	Self-Correct (round 2)	5	89.0	80.0	43.0

Reported Improvements need Oracle answers

		GSM8K	CommonSenseQA	HotpotQA
GPT-3.5	Standard Prompting	75.9	75.8	26.0
	Self-Correct (Oracle)	84.3	89.7	29.0
GPT-4	Standard Prompting	95.5	82.0	49.0
	Self-Correct (Oracle)	97.5	85.5	59.0

Oracle: Let LLMs self correct only when the answer is wrong

Luckily, some applications do have an Oracle available and can be leveraged !

Self-debug naturally leverage unit tests as oracle

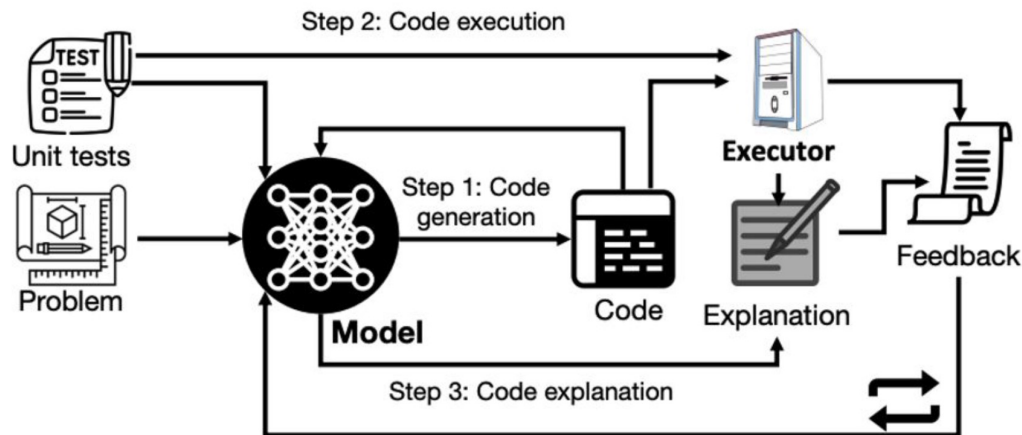


Figure 1: SELF-DEBUGGING for iterative debugging using a large language model. At each debugging step, the model first generates new code, then the code is executed and the model explains the code. The code explanation along with the execution results constitute the feedback message, which is then sent back to the model to perform more debugging steps. When unit tests are not available, the feedback can be purely based on code explanation.

How to Teach LLMs to Reason ?

How to “teach” / lead / force LLMs to Reason ?

- ❖ Type 1 Approaches: Using some Prompting techniques for the LLM, e.g. Chain-of-Thought (CoT) prompting
- ❖ Type 2 Approaches: Make LLM perform Search and Selection from Multiple Candidates
 - => Increase the **WIDTH** to explore solution space
- ❖ Type 3 Approaches: Make LLM perform Iterative self-improvement
 - => Increase the **DEPTH** to reach the final solution

Common Theme:

Make / Lead / Force LLM to use more Token Budget (i.e. “Think” more) before reaching the final solution !

i.e. Trigger/ Make the LLM to perform “System 2” like behavior of Human!

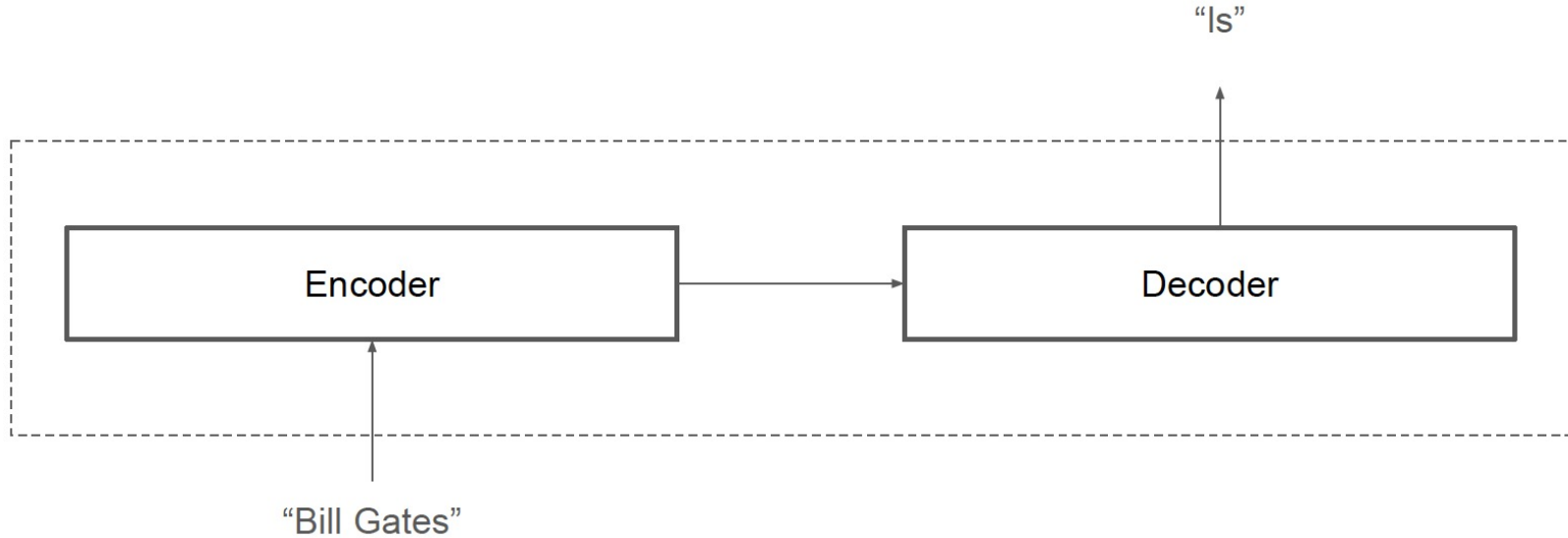
Type 1 Approaches: Teach LLMs to Reason via Prompting

Start with a Simple, Small Example – Last Letter Concatenation

Input	Output
“Elon Musk”	“nk”
“Bill Gates”	“ls”
“Barack Obama”	“?”

❖ Rule: Take the last letter of each word and concatenate them to form the output

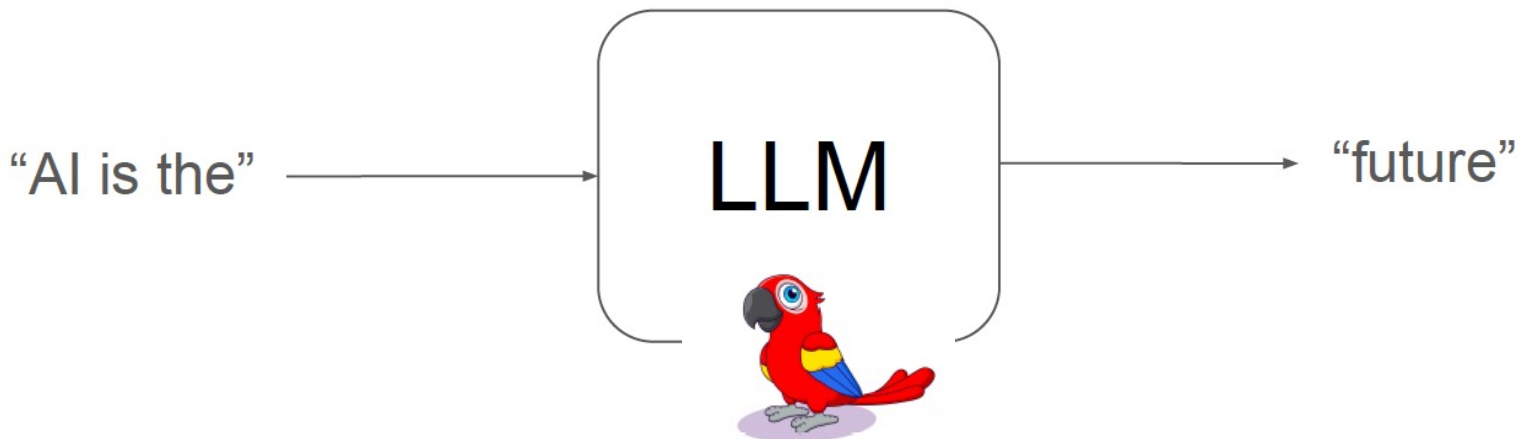
Attempt to solve it by Machine Learning / Deep Learning,
say, using a Seq2Seq Encoder-Decoder ? Needs Tons of Labelled Data !



- ❖ Can hardly call it “AI” as it requires vast amount of labelled data and training to do such a simple task !
- ❖ How can this problem be solved by an LLM ?

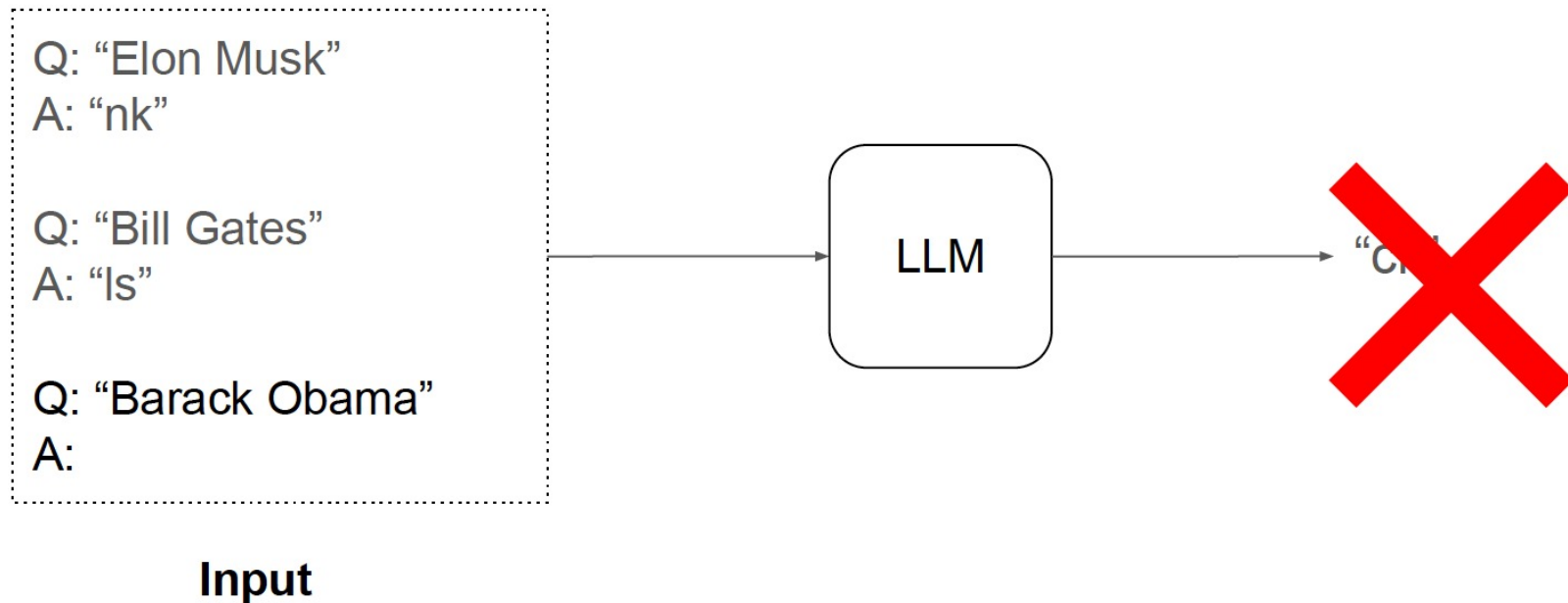
Recall: What an LLM is, What does it do ?

- ❖ LLM is a Transformer model trained to predict the Next Word, e.g.
“AI is the future”
- ❖ After trained with many sentences, e.g. text from the Internet, we can have:



- ❖ More like training parrots to mimic human language !

Few-shot Prompting for Last-Letter-Concatenation **DID NOT WORK !**



Sources:

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

T.B. Brown et al., Language models are few-shot learners, 2020, arXiv preprint arXiv:2005.14165.

Let's add “reasoning process” before “answer”

Q: “Elon Musk”

A: the last letter of "Elon" is "n". the last letter of "Musk" is "k". Concatenating "n", "k" leads to "nk". so the output is "nk".

reasoning process

Q: “Bill Gates”

A: the last letter of "Bill" is "l". the last letter of "Gates" is "s". Concatenating "l", "s" leads to "ls". so the output is "ls".

Q: “Barack Obama”

A:

Let's add “reasoning process” before “answer”

Q: “Elon Musk”

A: the last letter of "Elon" is "n". the last letter of "Musk" is "k". Concatenating "n", "k" leads to "nk". so the output is "nk".

reasoning process

Q: “Bill Gates”

A: the last letter of "Bill" is "l". the last letter of "Gates" is "s". Concatenating "l", "s" leads to "ls". so the output is "ls".

Q: “Barack Obama”

A: the last letter of "Barack" is "k". the last letter of "Obama" is "a". Concatenating "k", "a" leads to "ka". so the output is "ka".

Actually, ONE demonstration is enough, like Humans !

Q: "Elon Musk"

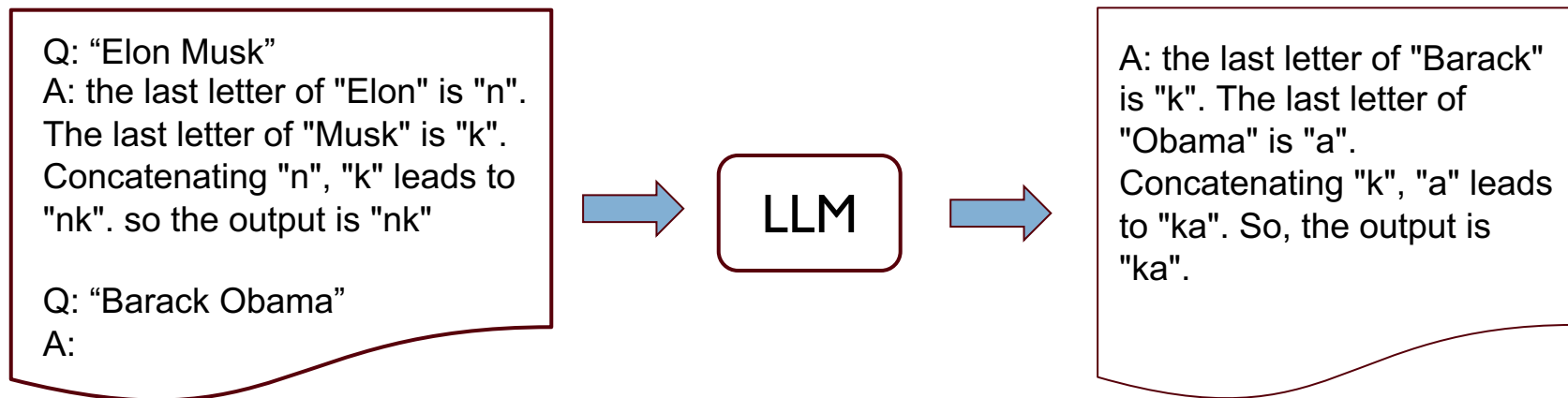
A: the last letter of "Elon" is "n". the last letter of "Musk" is "k". Concatenating "n", "k" leads to "nk". so the output is "nk".

Q: "Barack Obama"

A: the last letter of "Barack" is "k". the last letter of "Obama" is "a". Concatenating "k", "a" leads to "ka". so the output is "ka".

100% accuracy with only one demonstration example !

This approach is the so-called: Teaching LLM to Reason via Chain of Thought (CoT) Prompting



Key Idea (ingredient) behind the Chain-of-Thought approach:

Derive the Final Answer through **Intermediate Steps** !

[Ling et al 2017](#) in DeepMind pioneered using **natural language rationale** to solve math problems by “... **derive the final answer through a series of small steps**”. Trained a sequence-to-sequence model from scratch.

Problem 1:

Question: Two trains running in opposite directions cross a man standing on the platform in 27 seconds and 17 seconds respectively and they cross each other in 23 seconds. The ratio of their speeds is:

Options: A) $3/7$ B) $3/2$ C) $3/88$ D) $3/8$ E) $2/2$

Rationale: Let the speeds of the two trains be x m/sec and y m/sec respectively. Then, length of the first train = $27x$ meters, and length of the second train = $17y$ meters. $(27x + 17y) / (x + y) = 23 \rightarrow 27x + 17y = 23x + 23y \rightarrow 4x = 6y \rightarrow x/y = 3/2$

Correct Option: B



Ling et al. [Program Induction by Rationale Generation: Learning to Solve and Explain Algebraic Word Problems](#). ACL 2017

GSM8K: <Problem, Intermediate Steps, Answer>

Following the work by Ling et al 2017, Cobbe et al 2021 in OpenAI built a much larger math word problem dataset (GSM8K) with natural language rationales, and used it to finetune GPT3

Problem: Ali is a dean of a private school where he teaches one class. John is also a dean of a public school. John has two classes in his school. Each class has $\frac{1}{8}$ the capacity of Ali's class which has the capacity of 120 students. What is the combined capacity of both schools?

Solution: Ali's class has a capacity of 120 students. Each of John's classes has a capacity of $120/8 = 15$ students. The total capacity of John's two classes is $15 \text{ students} * 2 \text{ classes} = 30 \text{ students}$. The combined capacity of the two schools is $120 \text{ students} + 30 \text{ students} = 150 \text{ students}$.

Final answer: 150



Cobbe et al. Training Verifiers to Solve Math Word Problems. [arXiv:2110.14168](https://arxiv.org/abs/2110.14168) [cs.LG]. 2021

Show Your Work:

Scratchpads for Intermediate Computation with Language Models

Input:

2 9 + 5 7

Target:

<scratch>

2 9 + 5 7 , C: 0

2 + 5 , 6 C: 1 # added 9 + 7 = 6 carry 1

, 8 6 C: 0 # added 2 + 5 + 1 = 8 carry 0

0 8 6

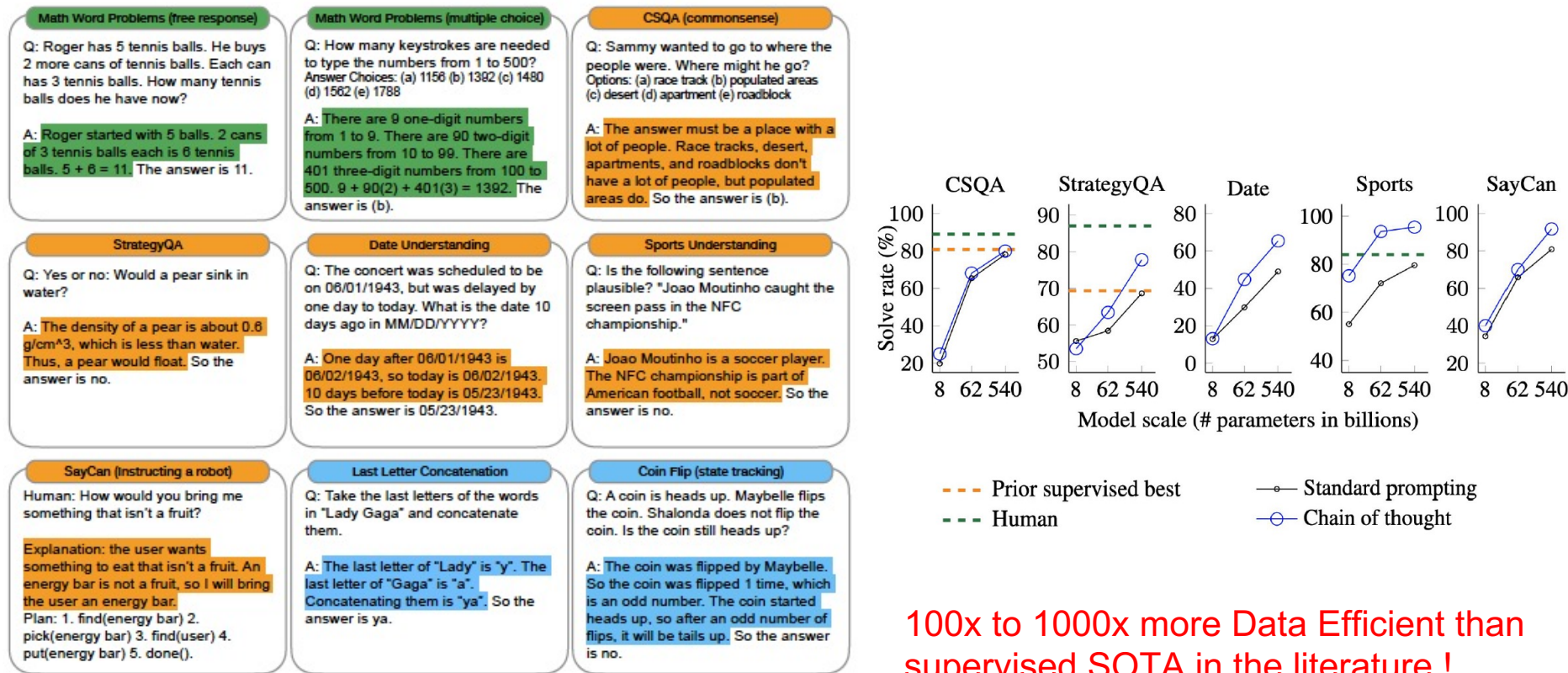
</scratch>

8 6



Nye et al. Show Your Work: Scratchpads for Intermediate Computation with Language Models. [arXiv:2112.00114](https://arxiv.org/abs/2112.00114) [cs.LG], 2021

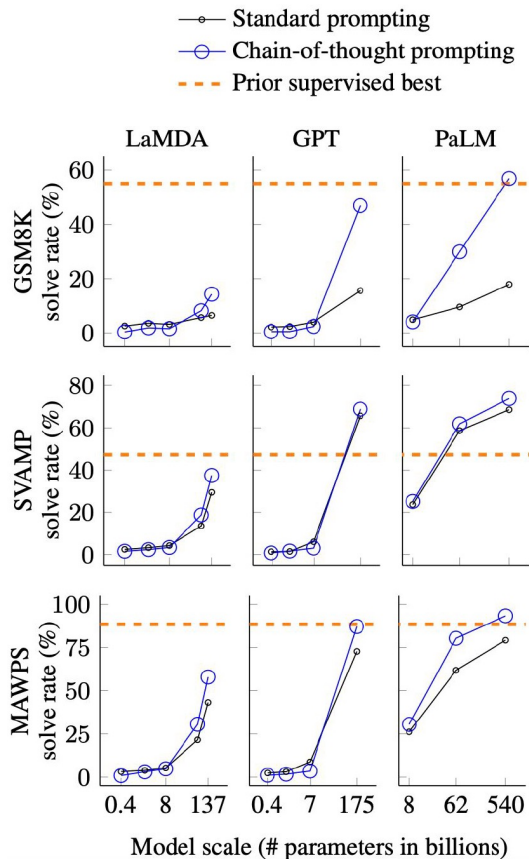
Chain-of-Thought (CoT) Prompting



100x to 1000x more Data Efficient than supervised SOTA in the literature !

Figure 3: Examples of (input, chain of thought, output) triples for arithmetic, commonsense, and symbolic reasoning benchmarks. Chains of thought are highlighted. Full prompts in Appendix G.

CoT Performance scales with LLM size



- CoT performance improves more significantly with the increase of the model size.
- Better models benefit more with CoT generation
 - A drastic improvement on reasoning performance when the model reaches a certain scale.
- Note: these experiments used pretrained-only LLMs
 - Recent post-trained LLMs might have different scaling curves, but the main conclusions still hold.

Source:

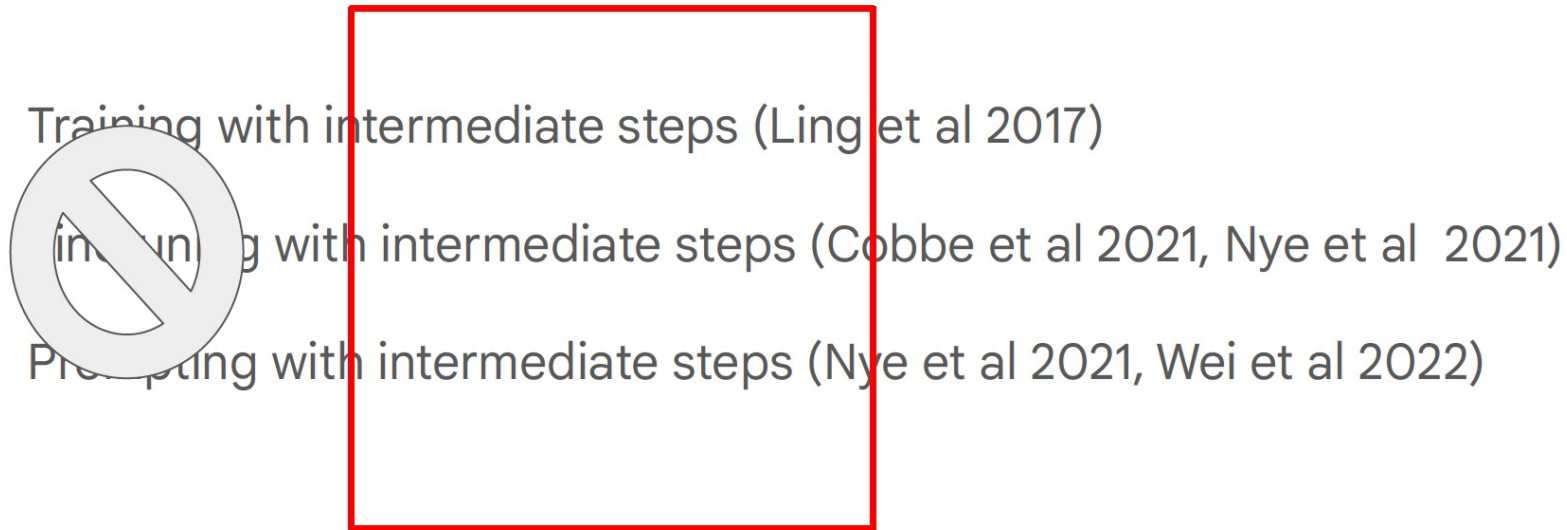
Jason Wei et al., [Chain-of-thought prompting elicits reasoning in large language models](#), NeurIPS 2022

Jason Wei et al., Emergent Abilities of Large Language Models, TMRL 2022.

Zero-shot CoT significantly outperforms Zero-shot

	Arithmetic					
	SingleEq	AddSub	MultiArith	GSM8K	AQUA	SVAMP
zero-shot	74.6/ 78.7	72.2/77.0	17.7/22.7	10.4/12.5	22.4/22.4	58.8/58.7
zero-shot-cot	78.0/78.7	69.6/74.7	78.7/79.3	40.7/40.5	33.5/31.9	62.1/63.7
	Common Sense		Other Reasoning Tasks		Symbolic Reasoning	
	Common SenseQA	Strategy QA	Date Understand	Shuffled Objects	Last Letter (4 words)	Coin Flip (4 times)
zero-shot	68.8/72.6	12.7/ 54.3	49.3/33.6	31.3/29.7	0.2/-	12.8/53.8
zero-shot-cot	64.6/64.0	54.8/52.3	67.5/61.8	52.4/52.9	57.6/-	91.4/87.8

Inclusion of **Intermediate Steps** is all that matters !



This is what really matters!

- ❖ Regardless of Training, Fine-Tuning or Prompting, when provided with examples that include **Intermediate Steps**, LLMs will generate responses that also include **intermediate steps** !

Why Intermediate Steps are helpful ?

Reasoning: Sufficient? Intermediate Guidance Helps...

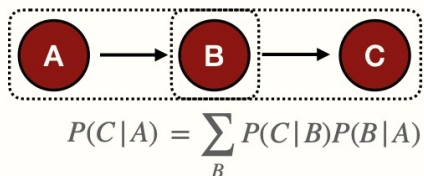
- ▶ Chain-of-thought (CoT) - series of intermediate reasoning steps
- ▶ Shown to improve LLM performance on complex reasoning tasks
- ▶ Inspired by human thought process: decompose multi-step problems
- ▶ Also provides an interpretable window into behavior of the model (how it arrived at an answer, where it goes wrong in its reasoning path)
- ▶ CoT exploits the fact that deep down in the model's weights, it knows more about the problem than just prompting it to get a response

Why Thinking Step-by-Step can help ?

- ❖ Why adding **Intermediate Steps** can help LLMs, which, after all, are merely **Statistical Inference models**, to be better in answering difficult questions ?

Hypothesis from B. Prystawski, M.Y. Li, N.D. Goodman:

Step-by-step reasoning lets (humans/LMs) chain together local inferences between variables they have seen together a lot in order to support longer-distance inferences



“Our results suggest that chain-of-thought reasoning is useful for language models because:

1) direct prediction is inaccurate for some inferences because the relevant variables are rarely seen together in training and

2) chain-of-thought reasoning improves estimation by incrementally chaining local statistical dependencies that are observed frequently in training.

We also find that the combination of locally structured training data and reasoning with self-generated intermediate variables yields much greater data efficiency than training on data containing all variables.”

References:

- Ben Prystawski, Michael Y. Li, Noah D. Goodman, “Why think step by step ? Reasoning emerges from the Locality of Experience,” NeurIPS 2023, <https://neurips.cc/virtual/2023/oral/73821>, <https://neurips.cc/media/neurips-2023/Slides/73821.pdf>.
- Noah Goodman, “Learning to Reason, Insights from Language Modeling,” MIT Center for Brains, Minds+ Machines (MITCBMM) seminar, <https://www.youtube.com/watch?v=YR9FztOF0R8>.
- S.C.Y. Chan et al., “Data Distributional Properties Drive Emergent In-Context Learning in Transformers,” NeurIPS 2022.

Thinking Step-by-Step due to Locality of Experience

Experiment Setup:

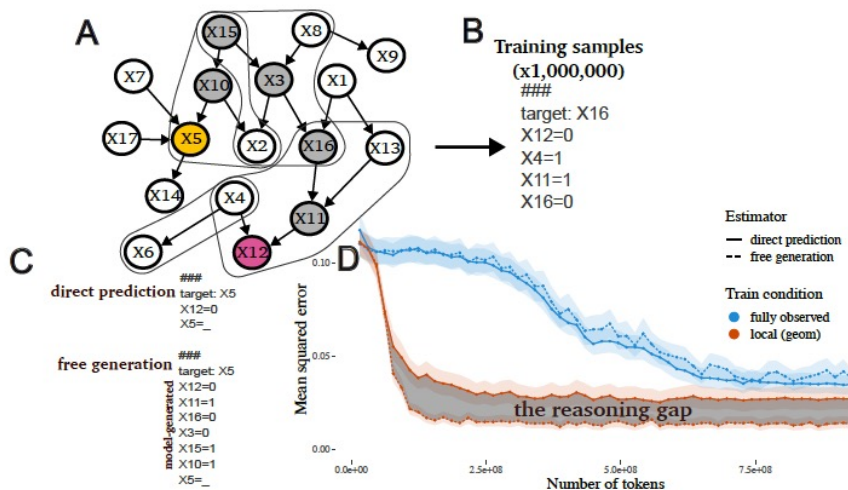


Figure 1: Overview of our training and estimation setup. A: visualization of a Bayes net. The pink variable is an example observed variable and the yellow variable is an example target variable. Gray variables are examples of useful intermediate variables for reasoning. Lines show examples of local neighborhoods from which training samples are drawn. B: format of the training samples. C: illustration of direct prediction and free generation estimators as prompts. We prompt the model to either immediately predict the target variable (direct prediction) or do so after generating intermediate variables and their values (free generation). We then compute mean squared errors between the estimated and true conditional probabilities. D: mean squared error by number of training tokens for each training condition and estimator. Ribbons indicate 95% confidence intervals.

References:

- Ben Prystawski, Michael Y. Li, Noah D. Goodman, “Why think step by step ? Reasoning emerges from the Locality of Experience,” NeurIPS 2023, <https://neurips.cc/virtual/2023/oral/73821>, <https://neurips.cc/media/neurips-2023/Slides/73821.pdf>.
- Noah Goodman, “Learning to Reason, Insights from Language Modeling,” MIT Center for Brains, Minds+ Machines (MITCBMM) seminar, <https://www.youtube.com/watch?v=Y99F-zt0E0R8>.

Why Intermediate Steps can help ?

Theoretical Results by Zhiyuan Li, Hong Liu, Denny Zhou, and Tengyu Ma, “Chain of Thought Empowers Transformers to Solve Inherently Serial Problems,” ICLR 2024

TL;DR: *We show both theoretically and empirically transformers with polynomial steps of CoT can simulate polysize circuits and thus are strictly more expressive than transformers without CoT.*

- ❖ **Transformer generating Intermediate Steps** can solve any Inherently serial problem as long as its depth exceeds a constant threshold
- ❖ **Transformer generating direct answers** either requires a huge depth to solve or cannot solve at all

Many Practical Implications of these Theorems

- ❖ Generating more Intermediate Steps
- ❖ Combining LLM with External Tools/ Search
- ❖ Diagnosing LLM failures and limitations
- ❖ And many more...

BUT Zero-shot CoT still performs worse than few-shot CoT

	MultiArith	GSM8K
Zero-Shot	17.7	10.4
Few-Shot (2 samples)	33.7	15.6
Few-Shot (8 samples)	33.8	15.6
Zero-Shot-CoT	78.7	40.7
Few-Shot-CoT (2 samples)	84.8	41.3
Few-Shot-CoT (4 samples : First) (*1)	89.2	-
Few-Shot-CoT (4 samples : Second) (*1)	90.5	-
Few-Shot-CoT (8 samples)	93.0	48.7
Zero-Plus-Few-Shot-CoT (8 samples) (*2)	92.8	51.5

How to improve CoT performance w/o manually labeling exemplars?

Next: Instruct LLM to generate exemplars itself via Analogical Prompting

- ❖ Prompt the LLM to first **RECALL** relevant exemplars, before solving the current problem

Pros:

- ❖ Exemplars are **self-generated by LLMs**, no manual crafting
- ❖ Exemplars are **tailored** to individual problems

Model Input

Q: What is the area of the square with the four vertices at (-2, 2), (2, -2), (-2, -6), and (-6, -2)?

Instruction:

Recall relevant exemplars:

Solve the initial problem:

Model Output

Relevant exemplars:

Q: What is the area of the square with a side length of 5?

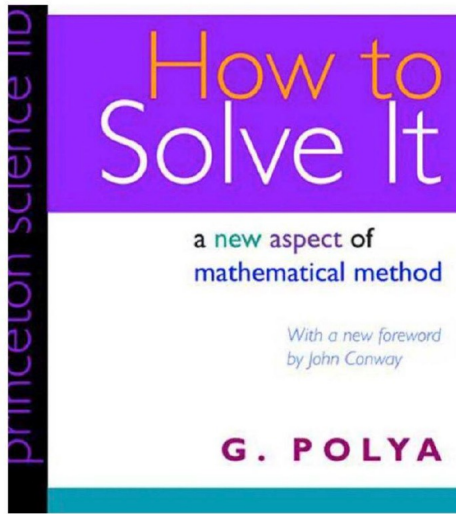
A: The area of a square is found by squaring the length of its side. So, the area of this square is $5^2 = 25$

Solve the initial problem:

To find the area of the square, we need to find the side length. The length is $\dots \sqrt{(2 - (-2))^2 + (-2 - 2)^2} = \sqrt{32}$. So, the area of the square is $(\sqrt{32})^2 = 32$.

Motivated by Human Analogical Reasoning

- ❖ Humans are not explicitly given demonstrations every time for a new task
- ❖ Instead, humans intrinsically recall from past relevant experience

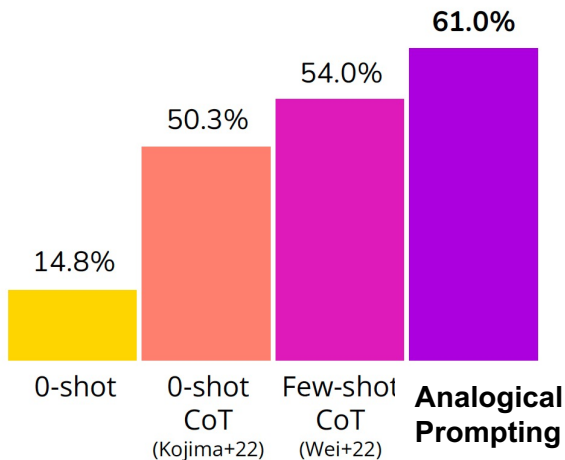


Do you know a related problem? We can scarcely imagine a problem absolutely new, unlike and unrelated to any formerly solved problem; but, if such a problem could exist, it would be insoluble. In fact, when solving a problem, we always profit from previously solved problems, using their result, or their method, or the experience we acquired solving them. And, of course, the problems from which we profit must be in some way related to our present problem. Hence the question: *Do you know a related problem?*

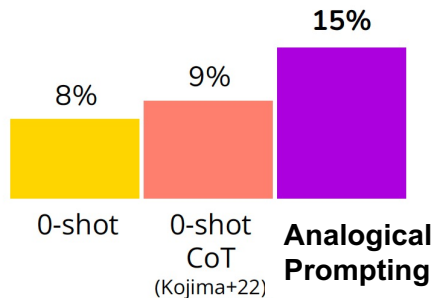
There is usually no difficulty at all in recalling formerly solved problems which are more or less related to our present one. On the contrary, we may find too many such problems and there may be difficulty in choosing a useful one. We have to look around for closely related problems; we LOOK AT THE UNKNOWN, or we look for a formerly solved problem which is linked to our present one by GENERALIZATION, SPECIALIZATION, or ANALOGY.

Analogical Prompting outperforms Zero-shot CoT & manual Few-shot CoT

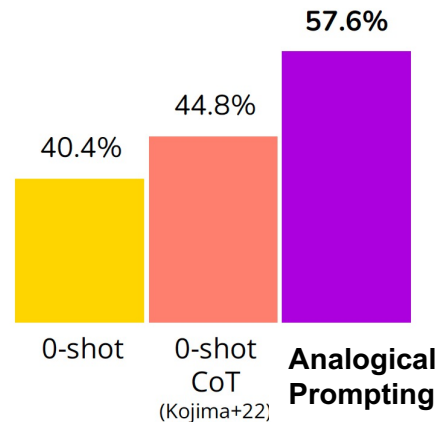
Math problems
(GSM8K, text-davinci-003)



Code generation
(Codeforces, GPT3.5 turbo)



Temporal reasoning
(BIG-Bench, GPT3.5 turbo)



Stronger LLMs are Better Analogical Reasoners

Prompting Method	(← scale down)			(scale up →)
	text-curie-001	text-davinci-001	text-davinci-002	text-davinci-003
0-shot	2%	6%	13%	14%
0-shot CoT	2%	6%	22%	50%
5-shot (fixed) CoT	2%	10%	43%	54%
5-shot retrieved CoT	3%	11%	47%	57%
Self-generated Exemplars	2%	9%	48%	61%

GSM8K for math reasoning

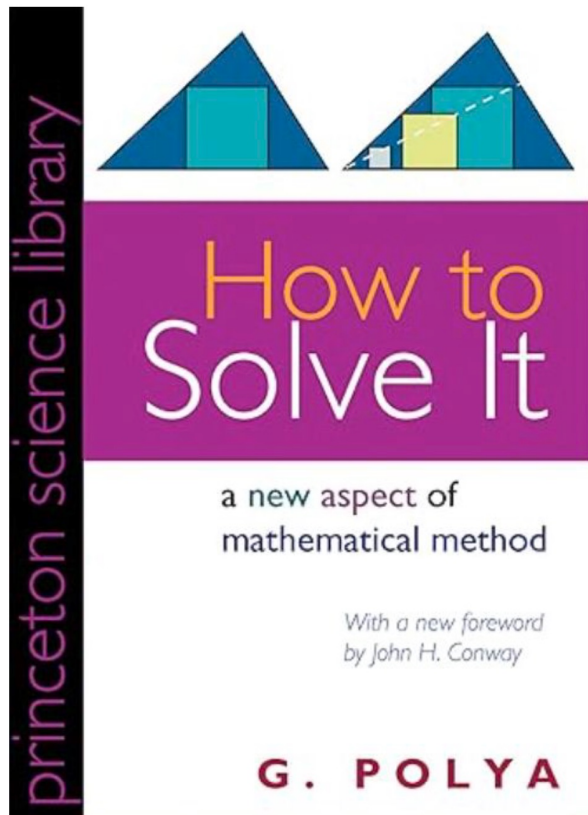
- ❖ Weaker LLMs benefit less from analogical prompting, though it does not hurt the zero-shot performance
- ❖ With stronger LLMs, analogical prompting outperforms CoT with manually-designed or retrieved exemplars
 - The generated CoT is more tailored to the underlying LLM

Other Strategies to improve Reasonings

- ❖ Recent samples that have been proposed and studied by DeepMind researchers:
 - ▶ Explicitly Instruct the LLM with the desired Reasoning Strategies for Problem Solving
 - ▶ Least-to-most Prompting: Easy-to-Hard Generalization via Decomposition [Zhou et al., ICLR2023]
 - ▶ Self-Discover: Instruct LLM to compose Reasoning Structures for each Task [Zhou et al., NeurIPS 2024]
 - ▶ Use LLMs for automated prompt-turning [Zhou et al., LLMs are Human-level Prompt Engineers, ICLR2023]
 - ▶ Chain-of-Thought Reasoning without Prompting (aka CoT decoding) [Wang, Zhou NeurIPS 2024] Empirical observations show some of the Top-K decoding paths (beyond the most probable one) actually contains / corresponds to valid CoT reasoning paths. One can develop an algorithm to “sift” through the Top-K decoding paths and isolate the most reliable paths for model output.

Least-to-most Prompting: Easy-to-Hard Generalization via Decomposition

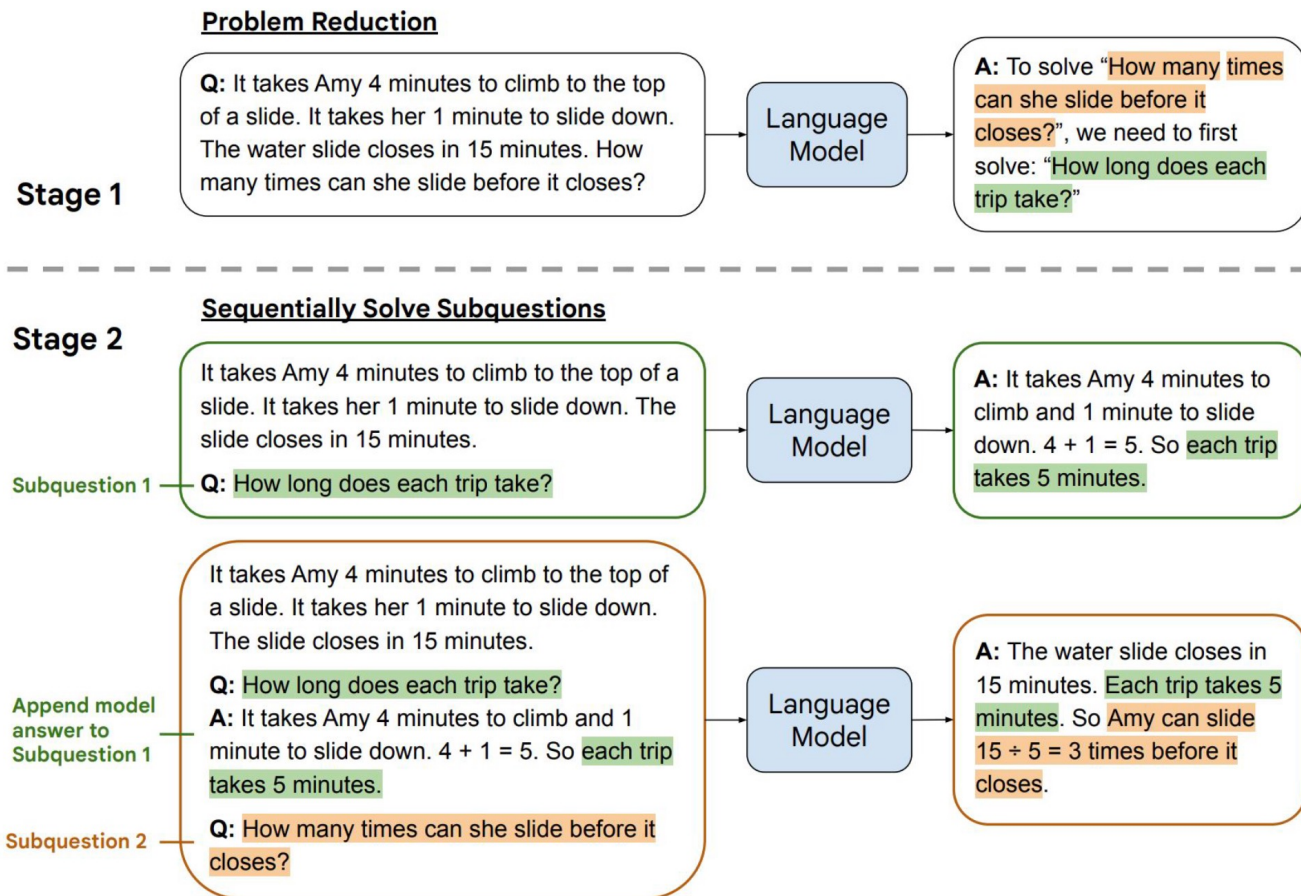
Decomposing and recombining are important operations of the mind.



differently. You decompose the whole into its parts, and you recombine the parts into a more or less different whole.

1. If you go into detail you may lose yourself in details. Too many or too minute particulars are a burden on the mind. They may prevent you from giving sufficient attention to the main point, or even from seeing the main point at all. Think of the man who cannot see the forest for the trees.

Least-to-most Prompting: Easy-to-Hard Generalization via Decomposition



Example: Solving the SCAN compositional generalization benchmark

Command	Action Sequence
“look thrice after jump”	JUMP LOOK LOOK LOOK
“run left and walk”	TURN_LEFT RUN WALK
“look opposite right”	TURN_RIGHT TURN_RIGHT LOOK

Method	Standard prompting	Chain-of-Thought	Least-to-Most
code-davinci-002	16.7	16.2	99.7
text-davinci-002	6.0	0.0	76.0
code-davinci-001	0.4	0.0	60.7

SCAN length split

- SCAN: translating synthetic natural language commands into actions sequences
- Length split: action sequences in the test set are longer than training samples
- Least-to-most prompting can achieve nearly perfect test accuracy with 0.1% training samples as exemplars

CoT without Prompting via CoT-Decoding [Wang & Zhou, NeurIPS 2024]

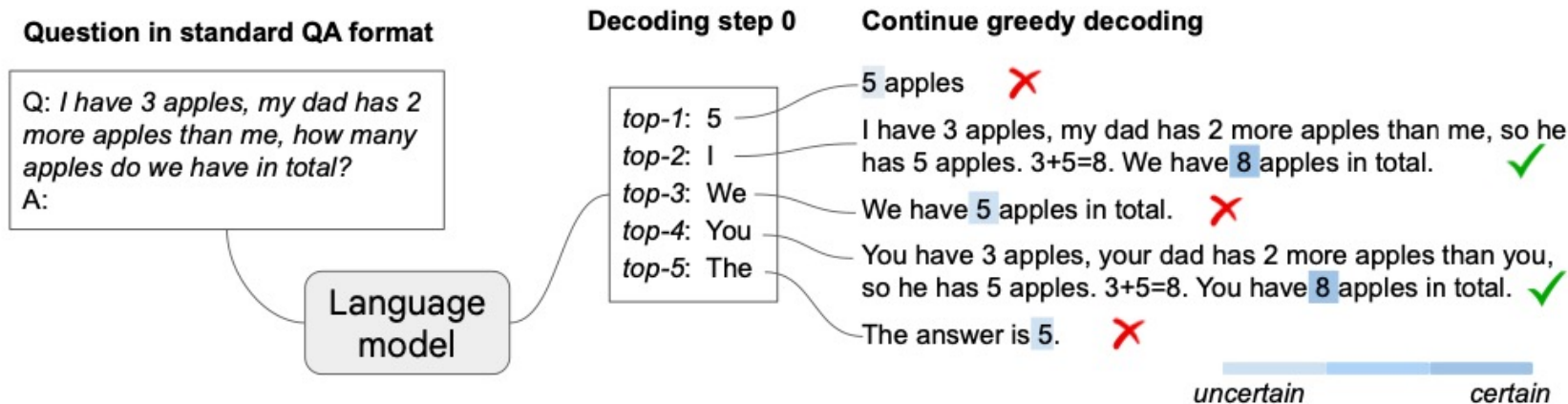


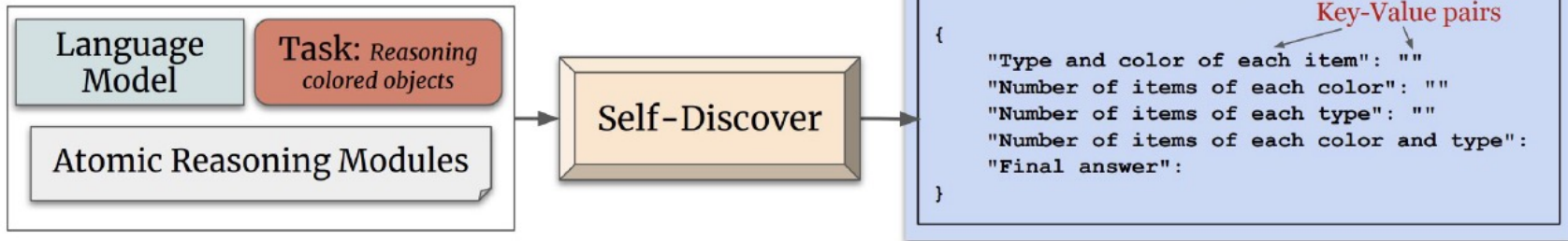
Figure 1 | Illustration of CoT-decoding. Pre-trained LLMs are capable of inherent reasoning without prompting by considering alternative top- k tokens, rather than solely relying on the top-1 greedy decoding path. Moreover, these models tend to display higher confidence in decoding the final answer (indicated by a darker shaded color) when a CoT reasoning path is present.

While most existing work suggest that LLMs falter in such direct-QA scenarios on reasoning (Cobbe et al., 2021a; Kojima et al., 2022; Nye et al., 2021; Wei et al., 2022), our findings reveal a nuanced picture. We observe that LLMs indeed struggle with reasoning when relying solely on greedily decoded paths. However, when we consider alternative paths among the top- k tokens, CoT reasoning patterns emerge naturally within the decoding trajectories of LLMs. In addition, we have observed an interesting pattern: the model demonstrates increased confidence in the final answer when a CoT reasoning path is present in the decoding process. As illustrated in Figure 1, this is evident where paths 2 and 4 show heightened certainty in arriving at the correct answer “8”, contrasting sharply with the high uncertainty in paths that lead to the incorrect “5”. Leveraging this phenomenon, we develop a method to sift through the top- k decoding paths, which we refer to as **CoT-decoding**, thereby isolating the most reliable paths for model output.

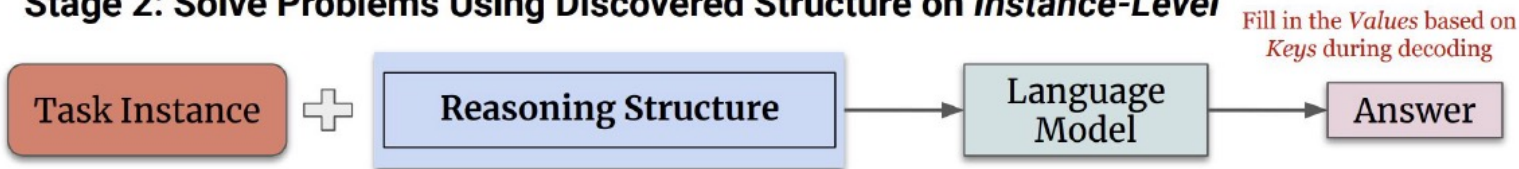
Self-Discover: Instruct LLM to compose Task-specific Reasoning Structures

[Zhou et al., NeurIPS 2024]

Stage 1: Discover Reasoning Structure on Task-Level



Stage 2: Solve Problems Using Discovered Structure on Instance-Level



- Different reasoning tasks require different reasoning structures, i.e., different ways to decompose the task and plan for each stage.
- Self-Discover composes task-specific reasoning structures without manually-written demonstrations.

Generalizing Chain-of-Thought Reasoning

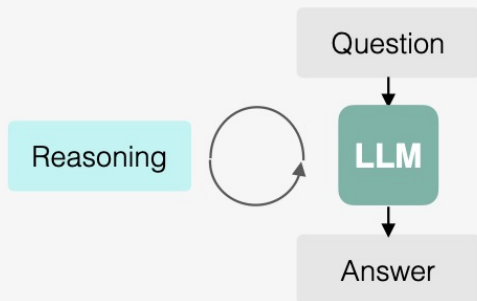
- ▶ Find CoT to have a more rigid definition and format
- ▶ Further, its advantages are for particular domains and types of questions
 - ▶ Task is challenging and requires multi-step reasoning
 - ▶ Scaling curve of the problem/task is relatively flat
- ▶ Humans think through different types of problems in multiple ways
- ▶ Our “scratchpad” is more flexible and open to different reasoning structures
- ▶ Maybe able to generalize CoT to be more flexible somehow

Generalization of the CoT approach

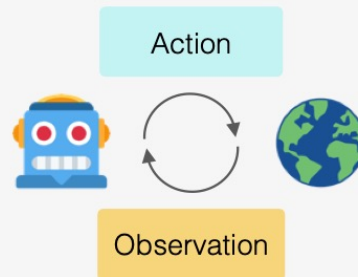
- ▶ Chain-of-thought (CoT) - series of intermediate reasoning steps
- ▶ Reasoning and Acting (ReAct)
- ▶ Tree-of-Thoughts

ReAct = Reasoning and Acting

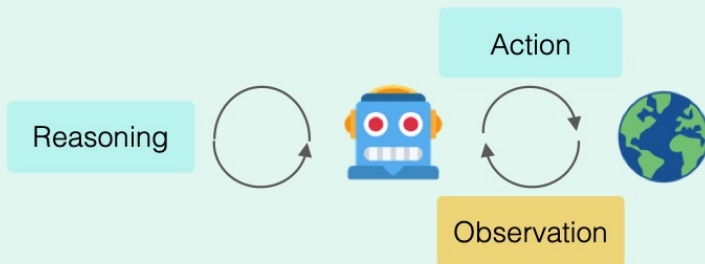
Reasoning (update internal belief)



Acting (obtain external feedback)



ReAct: a new paradigm of agents that **reason and act**



- **Synergy** of reasoning and acting
- **Simple** and intuitive to use
- **General** across domains

Example of ReAct w/ a Zero-Shot ReAct Prompt

You are an agent that answers questions by using two actions:

- `search[query]`: Google search the query. You can also use it to calculate math.
- `finish[answer]`: return the answer

Your generation should have the following format:

Thought: to analyze the observation and inform the action.

Action: your action

Question: If I have seven trillion dollars today, can I buy Apple, Nvidia, and Microsoft? If not, how much more money do I need?

Example of ReAct

LLM

(GPT-4)

Thought: I need to find the current market capitalization of Apple, Nvidia, and Microsoft to understand how much it would cost to buy all of them.

Action: search["current market capitalization of Apple, Nvidia, and Microsoft"]

Observation: "Nvidia's market cap stood at \$1.832 trillion as of about 10 a.m. ET Wednesday, while Alphabet's was at \$1.814 trillion, bringing Nvidia behind just Microsoft (MSFT) at \$3.015 trillion and Apple (AAPL) at \$2.836 trillion."



(Mar 2024)

Thought: Now that I have the market cap for all companies, let's add them together and see if \$7 trillion is enough.

Action: search["1.832 trillion + 3.015 trillion + 2.836 trillion"]

Observation: 7683000000000

Thought: The amount of money needed in addition to the \$7 trillion is \$683 billion.

Action: finish["You need an additional \$683 billion to be able to buy Apple, Nvidia, and Microsoft."] 

Example of ReAct

LLM

(GPT-4)

Thought: I need to find the current market capitalization of Apple, Nvidia, and Microsoft to understand how much it would cost to buy all of them.

Action: search["current market capitalization of Apple, Nvidia, and Microsoft"]

Observation: nothing found!

(My adversarial observation)

Thought: The search did not give any results. I will need to search for each company's market capitalization individually.

Action: search["current market capitalization of Apple"]

Observation: 'price': 180.75

(My adversarial observation)

Thought: This is the current price of Apple's stock, not its market capitalization. The market cap is calculated by multiplying the share price by the number of outstanding shares. Thus, I need to find the total number of Apple's outstanding shares to calculate its market cap.

Action: search["total number of Apple's outstanding shares"]



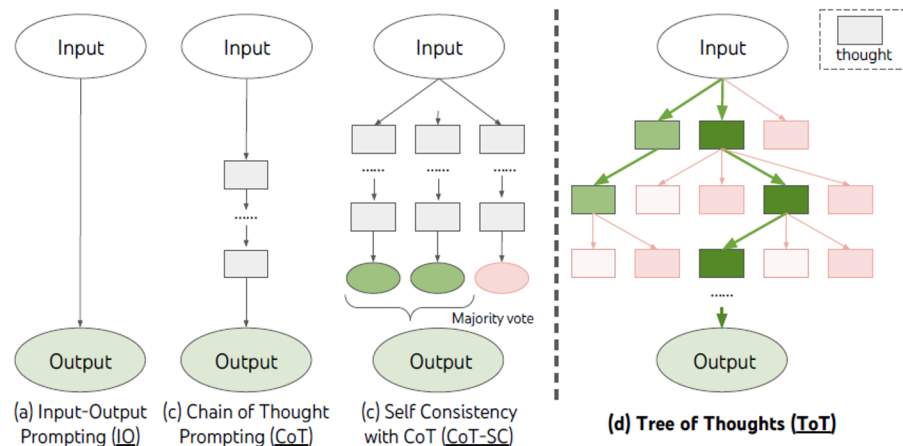
(Continues to solve the task)

Synergy: acting support reasoning, reasoning guides acting

Tree of Thoughts (System 2 like)

We want to “think”: convert time to accuracy.

- ToT: “consider multiple different reasoning paths and self-evaluating choices to decide the next course of action, as well as looking ahead or backtracking when necessary to make global choices” - [More details Later]



Like tree search in Chess, but in language.

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.

Summary so far: What CoT brings to LLM Reasoning ?

- ❖ Under the Chain-of-Thought approach, **Intermediate Steps** are added before the final model output is generated.
- ❖ Many ways to improve CoT performance at inference time:
 - ❖ Few-shot prompting with labeling of thoughts
 - ❖ Instruction prompting to trigger CoT generation
 - ❖ Instruct the LLM to use analogy to create relevant exemplars for the task on hand
 - ❖ Instruct the LLM to automate & optimize the prompt design
 - ❖ Instruct the LLM to apply decomposition techniques on the problem
 - ❖ CoT decoding

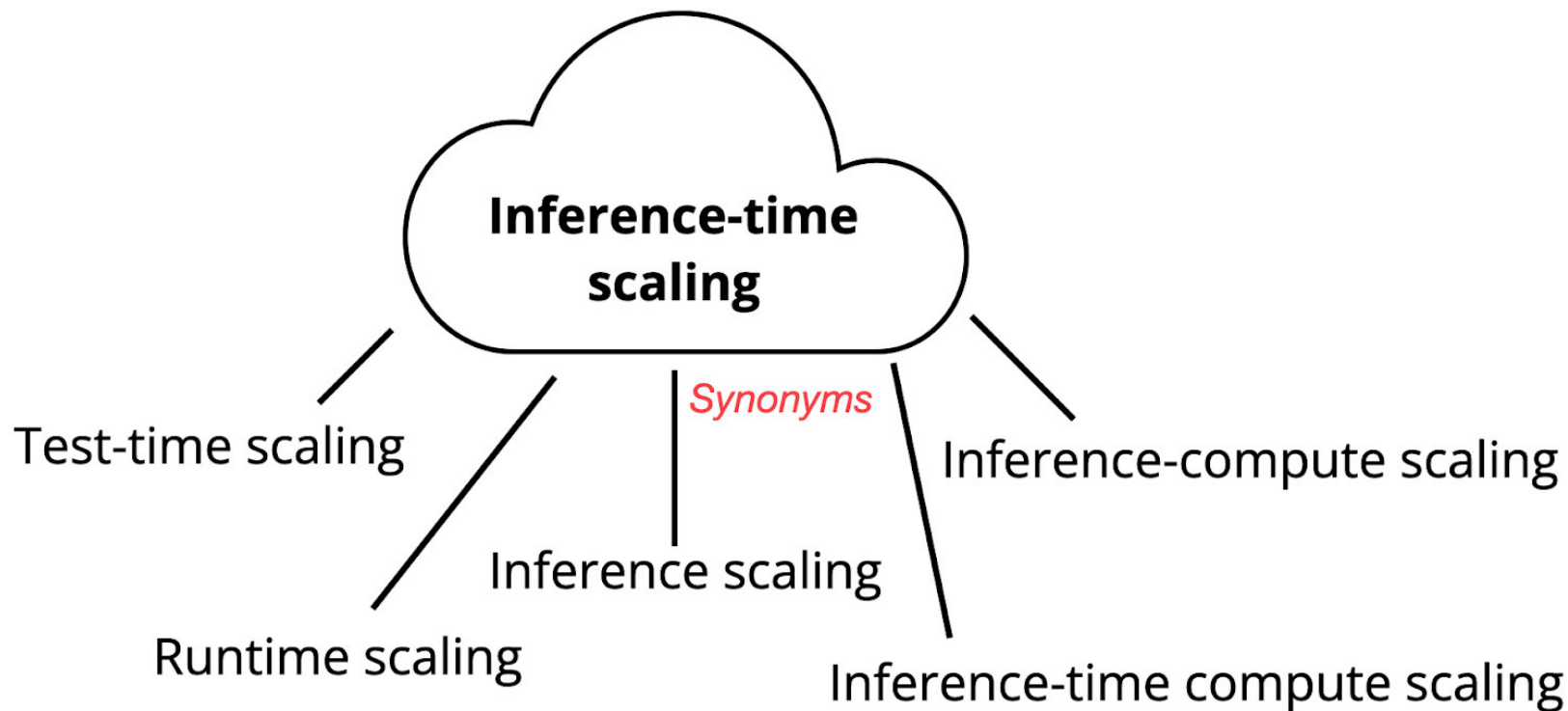
[Later: we will see how to improve the “reasoning” results by examining/ comparing MULTIPLE CoT paths!]

These techniques induce the use of more token budget to generate a single solution!

- ❖ CoT looks like Search / Planning in a Classical sense !
- ❖ In general, **More complex** questions => **Longer** CoT paths => **More** Reasoning steps
 - ⇒ CoT Prompting indirectly adjusts the amount of Computation (i.e. the “thinking process/ time” for the various intermediate steps) for tasks of different level of difficulties
 - ⇒ Kick-start the idea of “**Inference Time Scaling**” aka “**Test Time Scaling**” !

NB: Need some (can be more than one) mechanisms, e.g. Prompting, Supervised Fine-Tuning, Reinforcement Learning, Budget-forcing, etc, to induce CoT kind of behavior for a LLM !

The so-called “Inference-Time Scaling” - and its Synonyms



Type 2 Approaches:

Teach LLMs to Reason by expanding the **WIDTH** of the exploration space

Issues with Chain-of-Thought approach

- ❖ Generating intermediate steps (i.e. CoT approach) are helpful BUT are they producing the correct direct answers (i.e. being an unbiased estimator) ?
 - ▶ After all, LLMs is still (merely) a probabilistic, inference model predicting the next token !

What LLM does in decoding:

$$\arg \max \mathbb{P}(\text{reasoning path, final answer} | \text{problem})$$

What we want:

$$\arg \max \mathbb{P}(\text{final answer} | \text{problem})$$



To get the right answer, we need to take one-step further:

What we want:

$$\arg \max \mathbb{P}(\text{final answer}|\text{problem})$$

What LLM does in decoding:

$$\arg \max \mathbb{P}(\text{reasoning path, final answer}|\text{problem})$$

Luckily:

$$\begin{aligned} & \arg \max \mathbb{P}(\text{final answer}|\text{problem}) \\ &= \sum_{\text{reasoning path}} \mathbb{P}(\text{ reasoning path, final answer}|\text{problem}) \end{aligned}$$

Thus, we can just compute the sum !

How to compute the sum then ? Via Sampling !

But how to reduce Sampling Variance so that we can get the right answer faster ?

Asking LLM to generate multiple solutions for a problem before picking the final answer !

- ❖ Should NOT limit the LLM to generate only one solution per problem
- ❖ Tell LLM to explore, examine and compare multiple solutions which allows it to possibly recover from mistakes in a single generation (i.e. only a single CoT path).
Two different variants of this approach:
 - Generate **multiple candidate End-to-End solutions** per problem
 - Generate **multiple candidate “Next Steps”** given the current **partial** CoT path (i.e. “partial”, incomplete thought).

Challenge:

- ❖ How to select the **“Best”** choice among multiple candidates ?
 - In most cases, NO oracle scorer is available for the LLM at inference time

Improving Step-by-Step Reasoning via Self-Consistency Checks !

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, Denny Zhou.
[Self-Consistency Improves Chain of Thought Reasoning in Language Models](#). ICLR 2023.

Self-Consistency (SC): Select the Response w/ the most consistent final answer

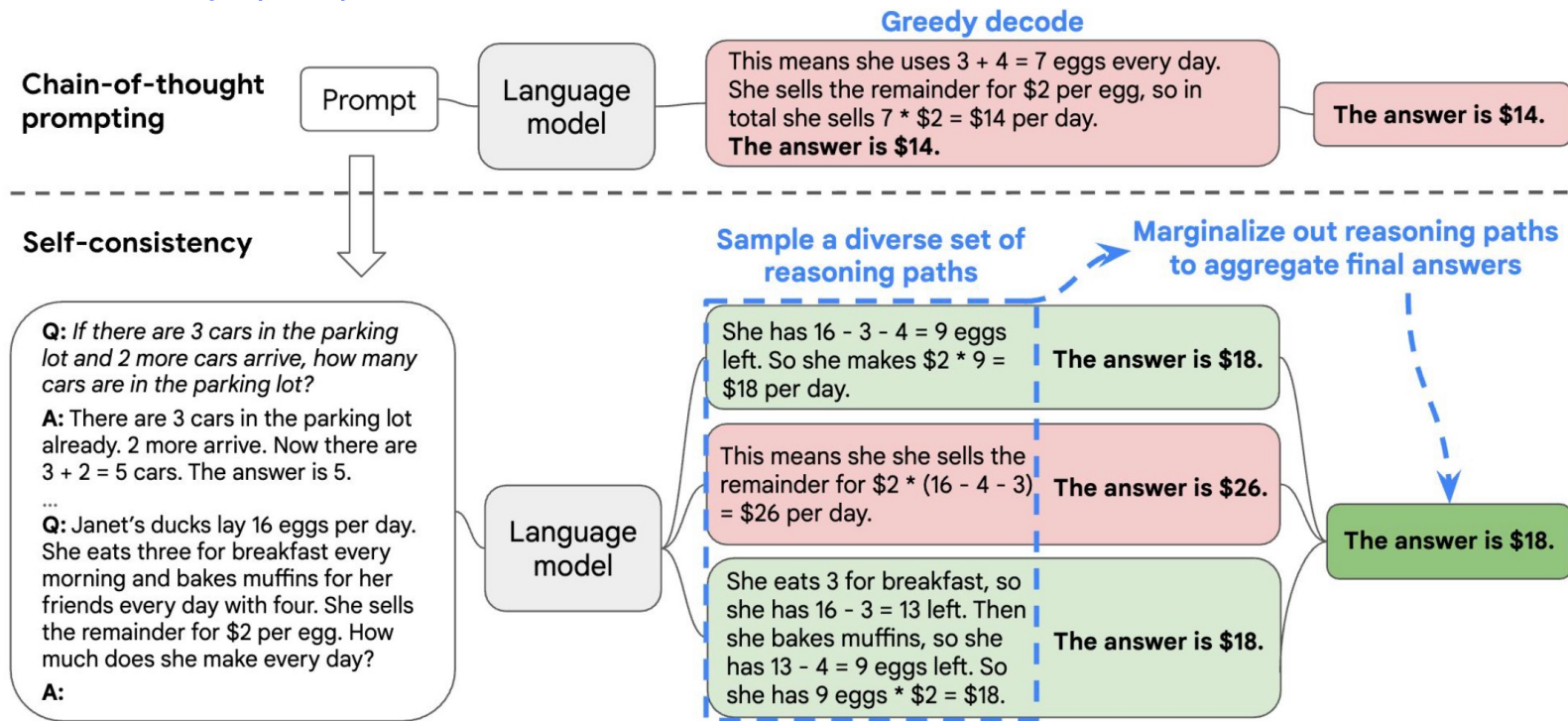


Figure 1: The self-consistency method contains three steps: (1) prompt a language model using example chains of thought; (2) sample from the language model's decoder to generate a diverse set of reasoning paths; and (3) choose the most consistent answer using the majority/plurality vote.

NB: The selection is only based on the final answer.

The reasoning paths do NOT need to be the same across different sampled responses

Self-Consistency (SC) Check

[Question] Janet's ducks lay 16 eggs per day. She eats three for breakfast every morning and bakes muffins for her friends every day with four. She sells the remainder for \$2 per egg. How much does she make every day?

Sampled responses:

Response 1: She has $16 - 3 - 4 = 9$ eggs left. So she makes $\$2 * 9 = \18 per day.

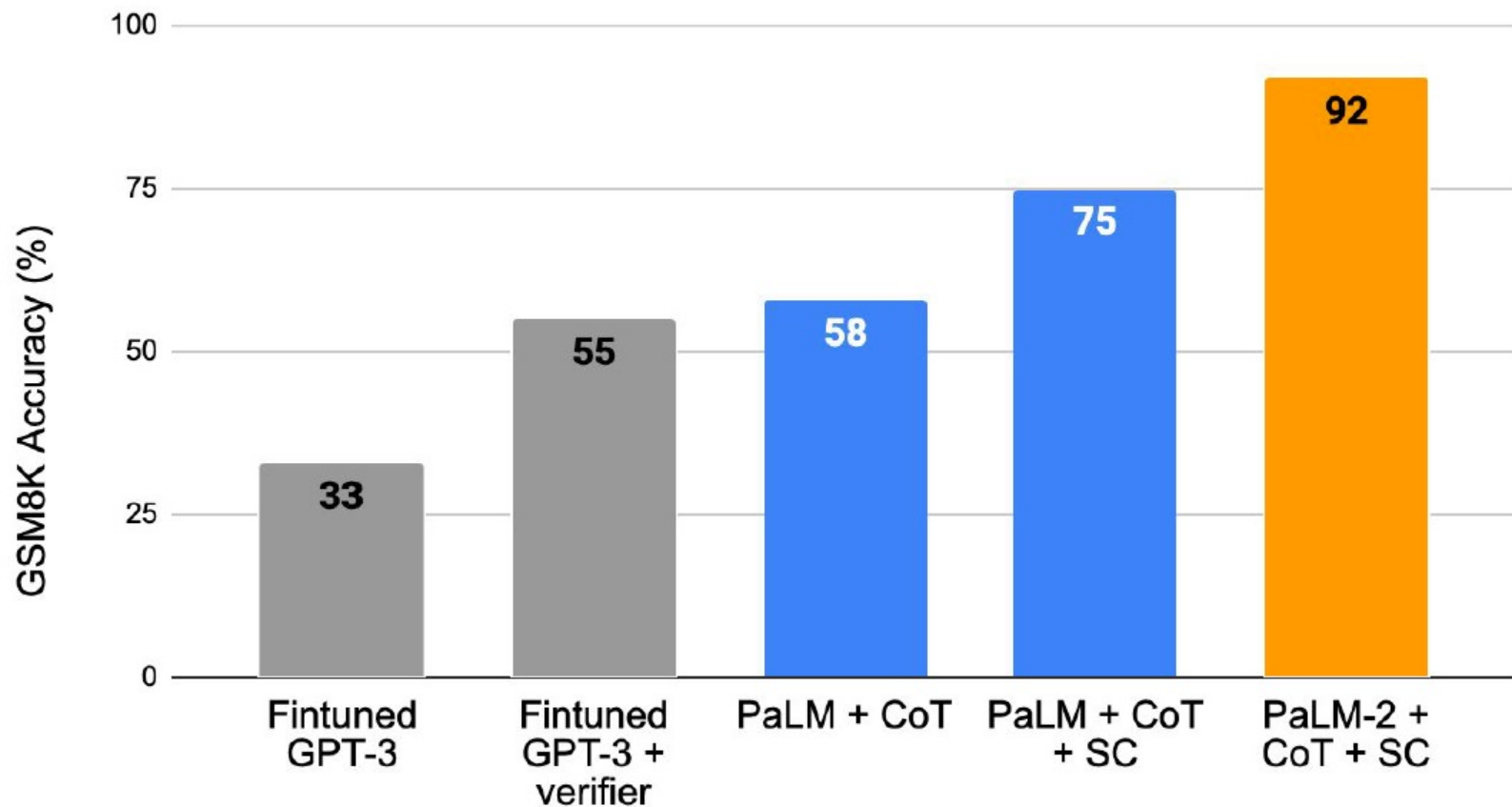
Response 2: This means she she sells the remainder for $\$2 * (16 - 4 - 3) = \26 per day.

Response 3: She eats 3 for breakfast, so she has $16 - 3 = 13$ left. Then she bakes muffins, so she has $13 - 4 = 9$ eggs left. So she has $9 \text{ eggs} * \$2 = \18 .

Most frequent answer is: 18
(Not most frequent reasoning path!)



GSM8K SOTA with only 8 examples



Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, Denny Zhou.
[Self-Consistency Improves Chain of Thought Reasoning in Language Models](#). ICLR 2023.

How many more examples are needed for Fine-tuning to be comparable to CoT+SC ?

“We perform finetuning... GSM8K ... it appears likely that the 175B model would require **at least two additional orders of magnitude** of training data to reach an **80% solve rate**.”

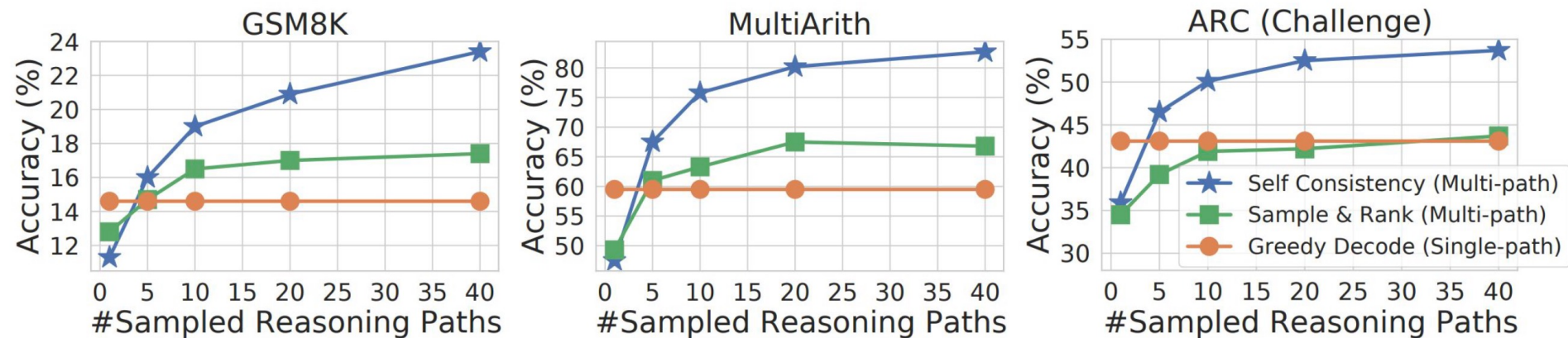
More Benchmark Results of using CoT+SC to Solve High-School Math problems

	Method	AddSub	MultiArith	ASDiv	AQuA	SVAMP	GSM8K
	Previous SoTA	94.9^a	60.5 ^a	75.3 ^b	37.9 ^c	57.4 ^d	35 ^e / 55 ^g
UL2-20B	CoT-prompting	18.2	10.7	16.9	23.6	12.6	4.1
	Self-consistency	24.8 (+6.6)	15.0 (+4.3)	21.5 (+4.6)	26.9 (+3.3)	19.4 (+6.8)	7.3 (+3.2)
LaMDA-137B	CoT-prompting	52.9	51.8	49.0	17.7	38.9	17.1
	Self-consistency	63.5 (+10.6)	75.7 (+23.9)	58.2 (+9.2)	26.8 (+9.1)	53.3 (+14.4)	27.7 (+10.6)
PaLM-540B	CoT-prompting	91.9	94.7	74.0	35.8	79.0	56.5
	Self-consistency	93.7 (+1.8)	99.3 (+4.6)	81.9 (+7.9)	48.3 (+12.5)	86.6 (+7.6)	74.4 (+17.9)
GPT-3 Code-davinci-001	CoT-prompting	57.2	59.5	52.7	18.9	39.8	14.6
	Self-consistency	67.8 (+10.6)	82.7 (+23.2)	61.9 (+9.2)	25.6 (+6.7)	54.5 (+14.7)	23.4 (+8.8)
GPT-3 Code-davinci-002	CoT-prompting	89.4	96.2	80.1	39.8	75.8	60.1
	Self-consistency	91.6 (+2.2)	100.0 (+3.8)	87.8 (+7.6)	52.0 (+12.2)	86.8 (+11.0)	78.0 (+17.9)

Lewkowycz et al., 2022. Solving Quantitative Reasoning Problems With Language Models.

Xuezhi Wang et al., Self-Consistency Improves Chain of Thought Reasoning in Language Models. ICLR 2023.

SC performance scales with more samples



- Sample-and-Rank baseline: select the response with the highest log probability
- Self-Consistency performance scales much better than probability-based ranking
 - Unless the model is trained to be a good verifier (will cover later)

Sampling Diverse Responses is crucial for good SC performance

	GSM8K	MultiArith	SVAMP	ARC-e	ARC-c
CoT (Wei et al., 2022)	17.1	51.8	38.9	75.3	55.1
Ensemble (3 sets of prompts)	18.6 $\pm$ 0.5	57.1 $\pm$ 0.7	42.1 $\pm$ 0.6	76.6 $\pm$ 0.1	57.0 $\pm$ 0.2
Ensemble (40 prompt permutations)	19.2 $\pm$ 0.1	60.9 $\pm$ 0.2	42.7 $\pm$ 0.1	76.9 $\pm$ 0.1	57.0 $\pm$ 0.1
Self-Consistency (40 sampled paths)	27.7 $\pm$ 0.2	75.7 $\pm$ 0.3	53.3 $\pm$ 0.2	79.3 $\pm$ 0.3	59.8 $\pm$ 0.2

Beam size / Self-consistency paths		1	5	10	20	40
AQuA	Beam search decoding (top beam)	23.6	19.3	16.1	15.0	10.2
	Self-consistency using beam search	23.6	19.8 $\pm$ 0.3	21.2 $\pm$ 0.7	24.6 $\pm$ 0.4	24.2 $\pm$ 0.5
	Self-consistency using sampling	19.7 $\pm$ 2.5	24.9 $\pm$ 2.6	25.3 $\pm$ 1.8	26.7 $\pm$ 1.0	26.9 $\pm$ 0.5
MultiArith	Beam search decoding (top beam)	10.7	12.0	11.3	11.0	10.5
	Self-consistency using beam search	10.7	11.8 $\pm$ 0.0	11.4 $\pm$ 0.1	12.3 $\pm$ 0.1	10.8 $\pm$ 0.1
	Self-consistency using sampling	9.5 $\pm$ 1.2	11.3 $\pm$ 1.2	12.3 $\pm$ 0.8	13.7 $\pm$ 0.9	14.7 $\pm$ 0.3

- ❖ Beam search: Keep Top-K paths with the highest probabilities in the Decoding process
- ❖ Ensemble baselines: apply greedy decoding for all prompt variants of a problem
- ❖ Self-Consistency (SC) uses sampling scales with more samples:
 - The sampling method needs to ensure response diversity, e.g., using a high temperature, nucleus sampling etc.

More Benchmark Results of using CoT+SC to Solve High-School Math problems

- Finetuning PaLM with math data
- SC + CoT solves **50%**!
- Non-math graduate students solve: 40%

Question: A particle moves so that it is at $(3\sin \frac{t}{4}, 3\cos \frac{t}{4})$ at time t . Find the speed of the particle, measured in unit of distance per unit of time.

Target: The parametric curve $(x, y) = (3\sin \frac{t}{4}, 3\cos \frac{t}{4})$ describes a circle with radius 3. Furthermore, it makes a full revolution at time $t = 8\pi$. (*figure*) Therefore, the speed of the particle is $\frac{6\pi}{8\pi} = \boxed{\frac{3}{4}}$.

Model output: The speed is the magnitude of the velocity vector. Therefore, we must first find the velocity vector, which is the derivative of the position vector. We have

$$\mathbf{v} = \frac{d}{dt} \left(3\sin \frac{t}{4}, 3\cos \frac{t}{4} \right) = 3 \left(\cos \frac{t}{4}, -\sin \frac{t}{4} \right) \frac{1}{4} = \left(\frac{3}{4} \cos \frac{t}{4}, -\frac{3}{4} \sin \frac{t}{4} \right).$$

Therefore, the magnitude of the velocity vector is

$$\begin{aligned} |\mathbf{v}| &= \sqrt{\left(\frac{3}{4} \cos \frac{t}{4} \right)^2 + \left(-\frac{3}{4} \sin \frac{t}{4} \right)^2} = \sqrt{\frac{9}{16} \cos^2 \frac{t}{4} + \frac{9}{16} \sin^2 \frac{t}{4}} \\ &= \frac{3}{4} \sqrt{\cos^2 \frac{t}{4} + \sin^2 \frac{t}{4}} = \frac{3}{4}. \end{aligned}$$

Therefore, the speed of the particle is $\boxed{\frac{3}{4}}$.

Why does Self-Consistency (SC) work ?

$$\begin{aligned} & \arg \max \mathbb{P}(\text{answer}|\text{problem}) && \text{(find the answer with the maximum probability)} \\ = & \sum_{\text{rationale}} \mathbb{P}(\text{answer}, \text{rationale}|\text{problem}) && \text{(sum over all latent reasoning paths)} \\ \approx & \frac{\text{frequency of the answer}}{\text{total number of sampled responses}} && \text{(approximate the sum by sampling)} \\ \propto & \text{frequency of the answer} && \text{(ignore the common factor of the same size)} \end{aligned}$$

Thus, $\arg \max \mathbb{P}(\text{answer}|\text{problem}) \approx$ selecting the most frequent answer.

Why does Self-Consistency (SC) work ?

Self-consistency is the empirical implementation of marginalization. Don't be superficial to interpret it as majority voting!



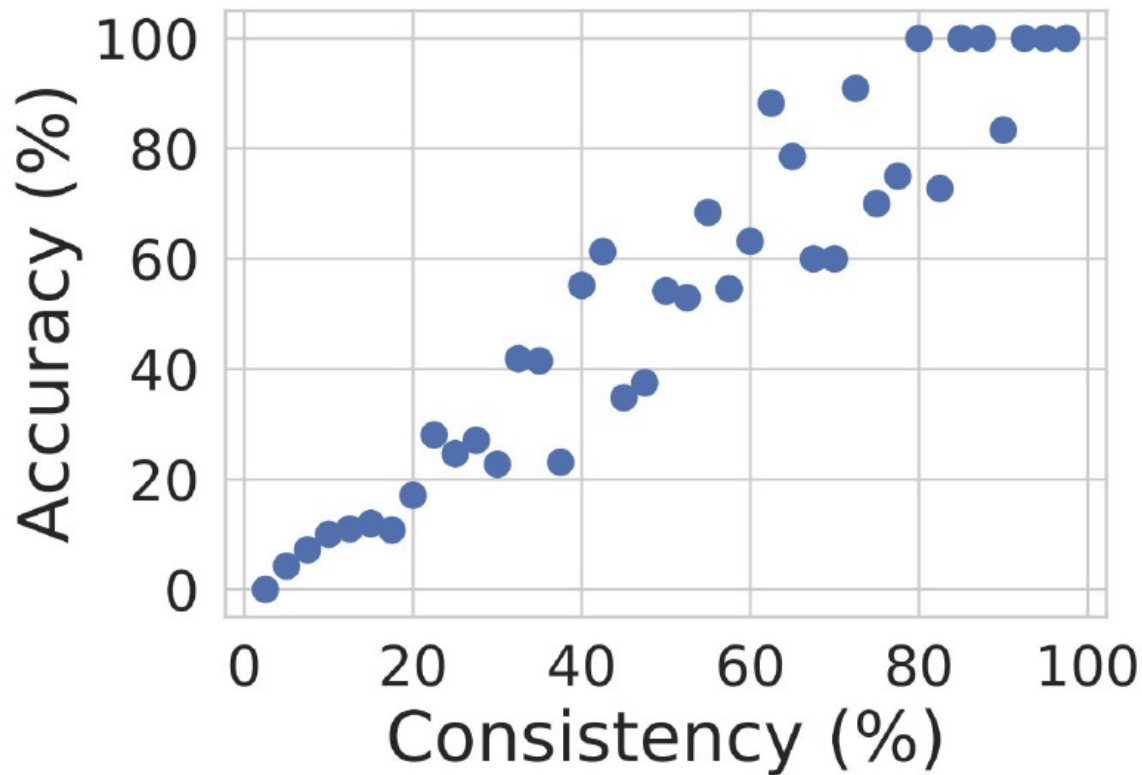
Self-consistency $\arg \max \mathbb{P}(\text{answer}|\text{problem})$

Chain-of-thought $\arg \max \mathbb{P}(\text{answer}, \text{rationale}|\text{problem})$

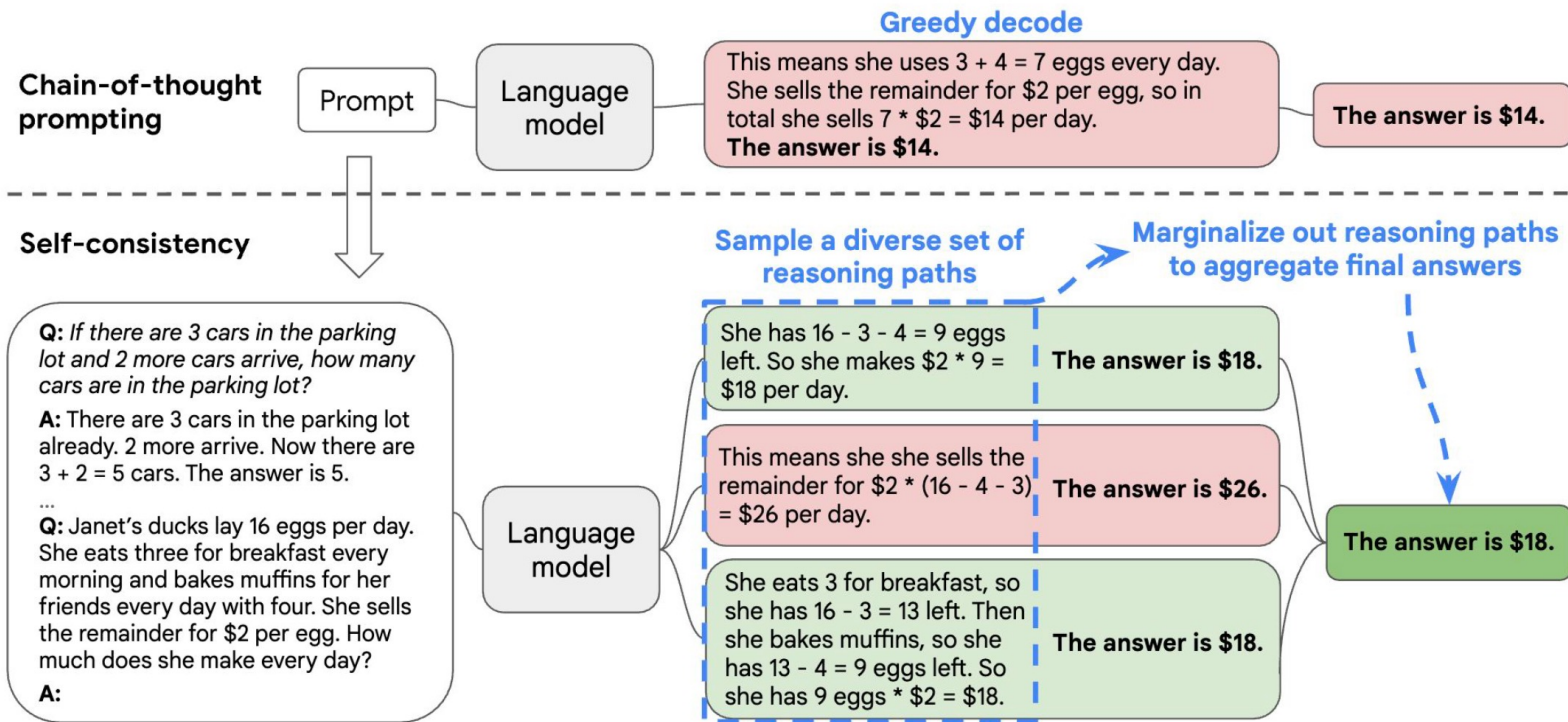
When there is no reasoning path, we don't need self-consistency, since we can then directly choose the most likely answer based on the given probabilities!



More consistent, more likely to be correct



Limitation of SC: require an answer extraction process



Can we enable consistency-based decoding for **free-form generation**?

How to apply Self-Consistency check on “free-form” answers ?

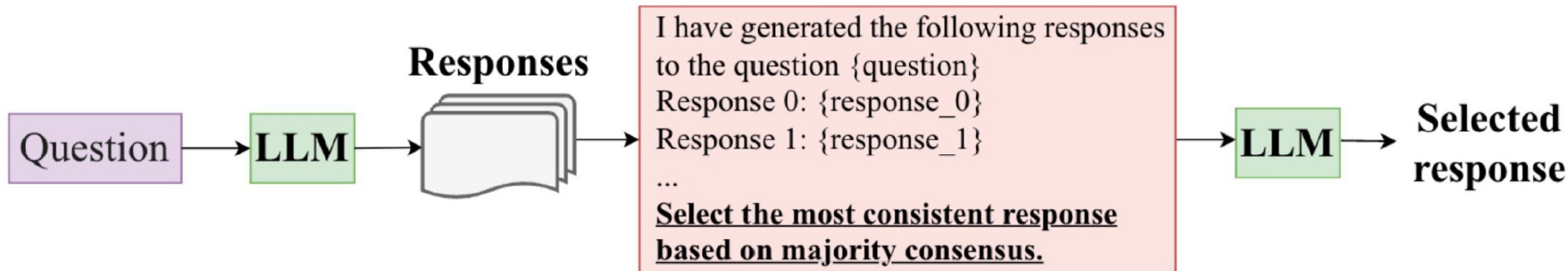
Universal Self-Consistency (USC)

Ask LLMs to self-select the most consistent answer

Universal Self-Consistency (USC):

Ask the LLM to perform consistency-based selection

Universal Self Consistency prompt



Universal Self-Consistency (USC) check

[Question] Where do people drink less coffee than they do in Mexico?

Response 1: ... Some examples include Japan, China and the United Kingdom.

It is important to note that coffee consumption can vary among individuals within these countries, and preferences can change depending on different factors such as...

Response 2: People in countries like Japan, China, and India typically drink less coffee than they do in Mexico...

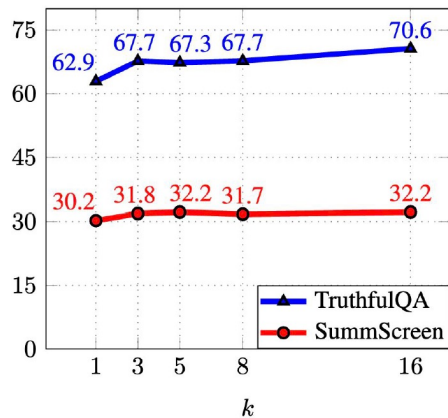
Response 3: There are several countries where people generally drink less coffee compared to Mexico. Some of these countries include:

1. Japan:...
2. China...
3. Saudi Arabia...
4. India...

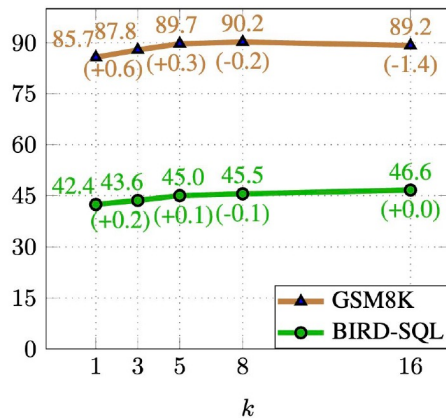
...

The most consistent response: 2

Performance of Universal Self-Consistency



(a) Results on open-ended generation.



(b) Results on GSM8K and BIRD-SQL. The top numbers are USC accuracies, and the bottom numbers are the differences to SC accuracies.

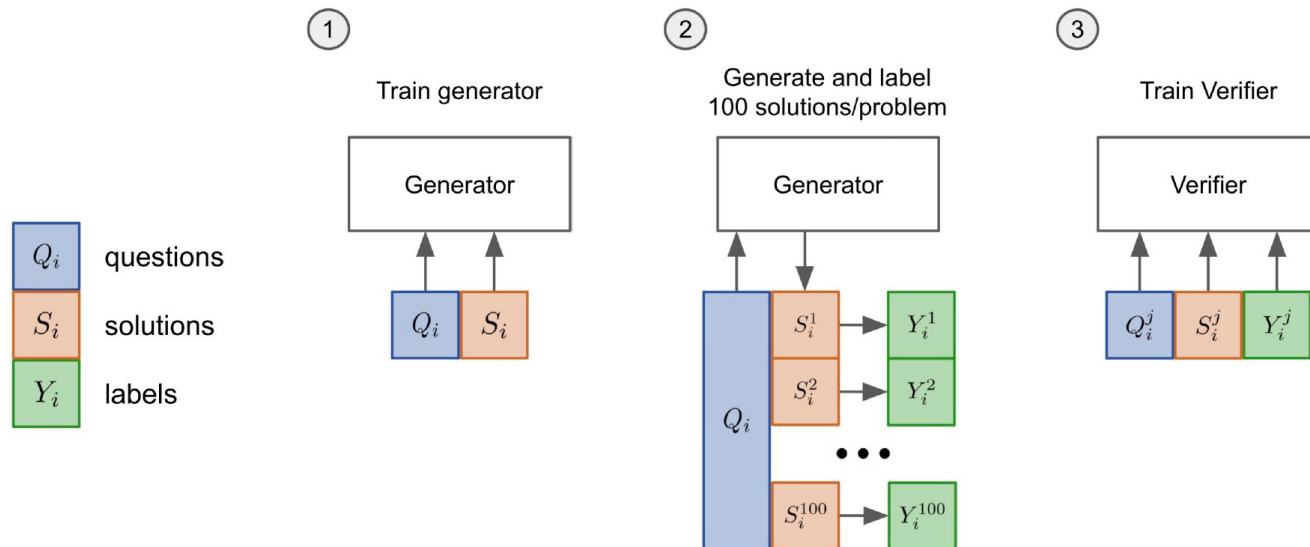
- Universal Self-Consistency (USC) improves the performance on open-ended generation (summarization, QA), where the original self-consistency is inapplicable.
- USC matches self-consistency performance on math reasoning and coding.
 - USC does not require answer extraction and code execution.
- USC performance is bounded by the long-context capability.

Best-of-N with Learned Verifier

STEPS:

1. Fine-Tune Verifier LM
2. Sample N answers
3. Select the Best answer according to the Verifier

Train LLM Rankers to further improve Consistency-based Selection



Two types of LLM-based Verifiers / Reward models:

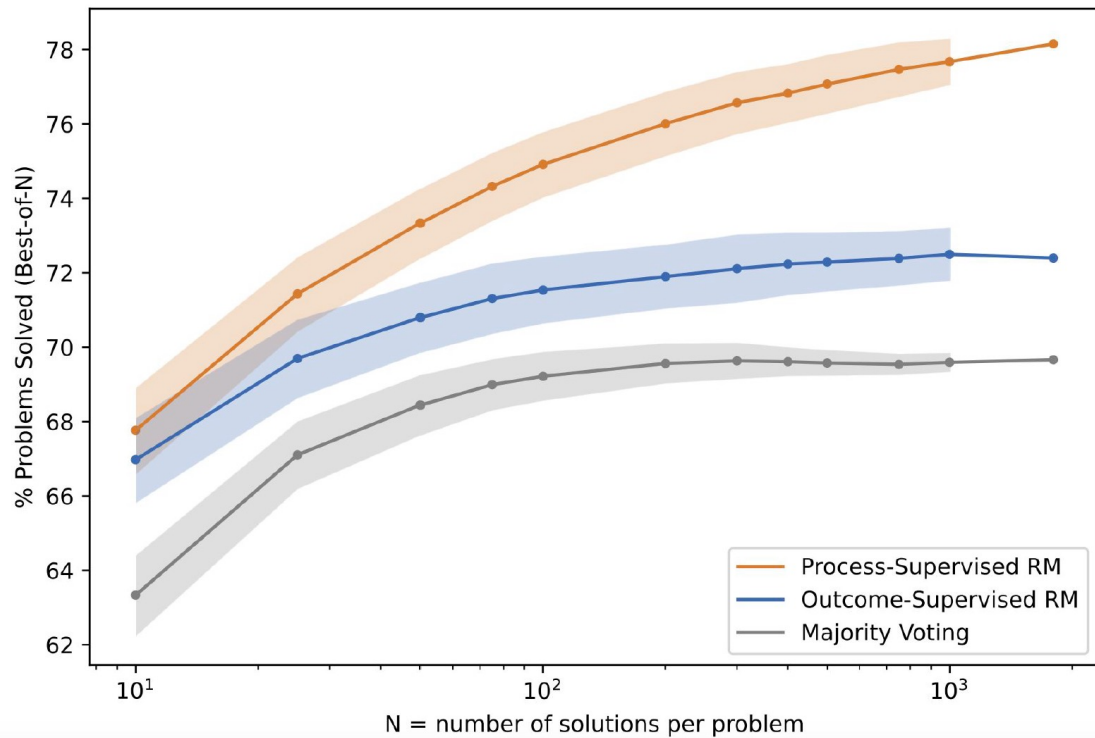
- ❖ Outcome-supervised Reward Model (ORM): Verify at the **(final) solution/ outcome** level
- ❖ Process-supervised Reward Model (PRM): Verify at the **Step-level** for each solution

References: Cobbe et al., Training Verifiers to Solve Math Word Problems, 2021.

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

(Strong) LLM-based Verifiers outperform Consistency-based selection

	ORM	PRM	Majority Voting
% Solved (Best-of-1860)	72.4	78.2	69.6



- ❖ Process-supervised Reward Model (PRM) scales better with more samples

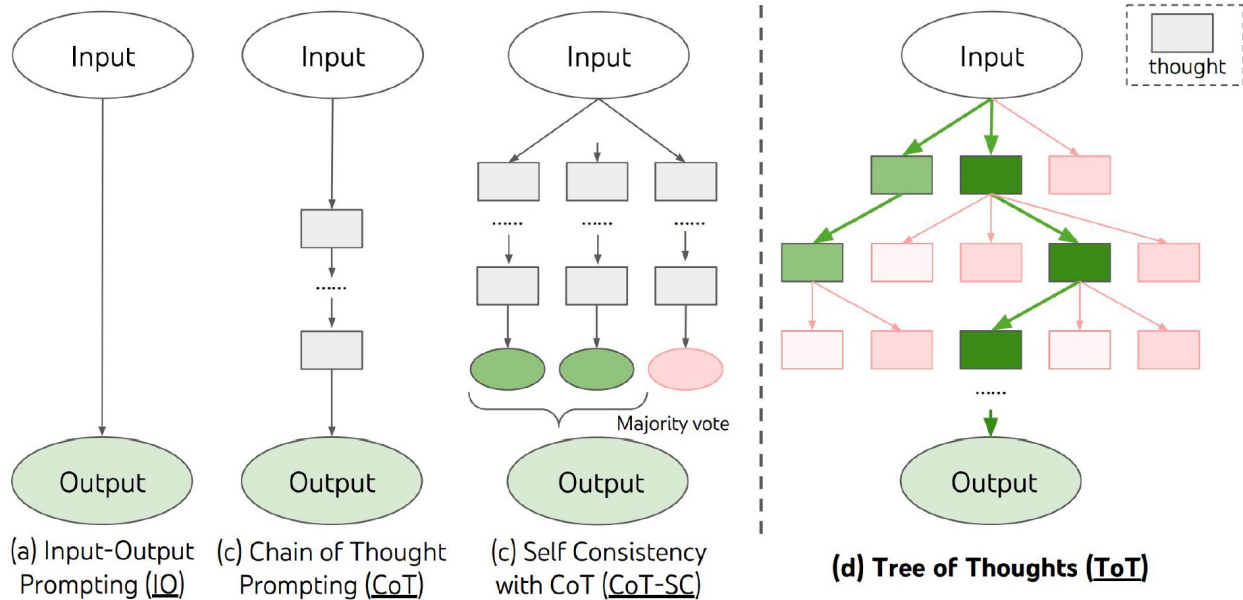
BUT:

- ❖ Performance highly dependent on verifier quality
- ❖ Same verifier may not generalize across different tasks

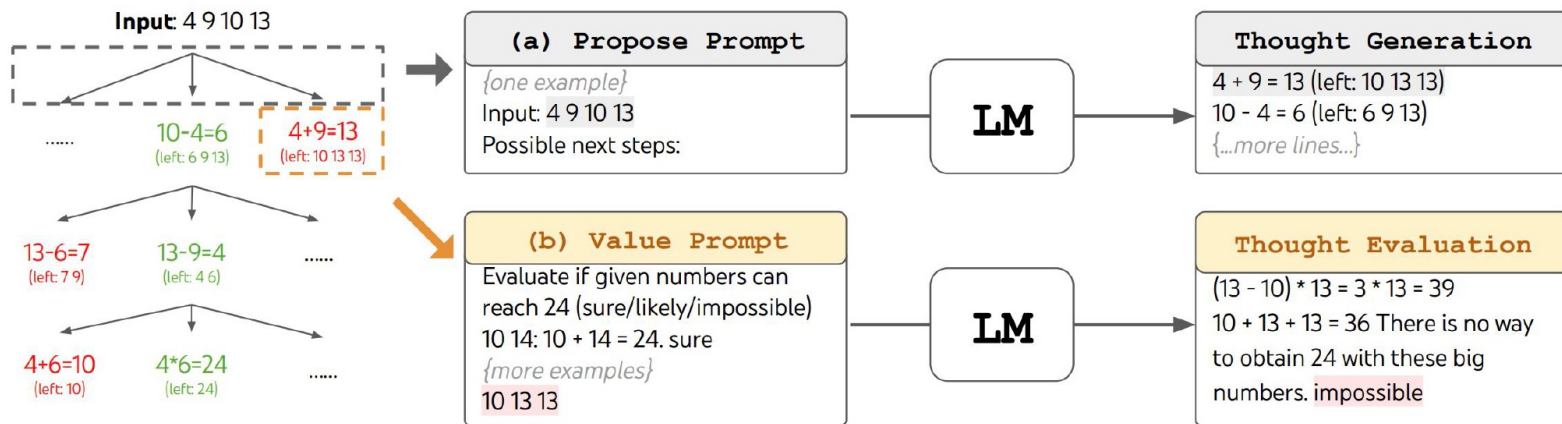
So far: Response Selection only after Full responses are generated

⇒ Does not fully utilize Step-wise scoring

❖ LLM + Tree search: Prioritize the exploration of more promising partial solutions !



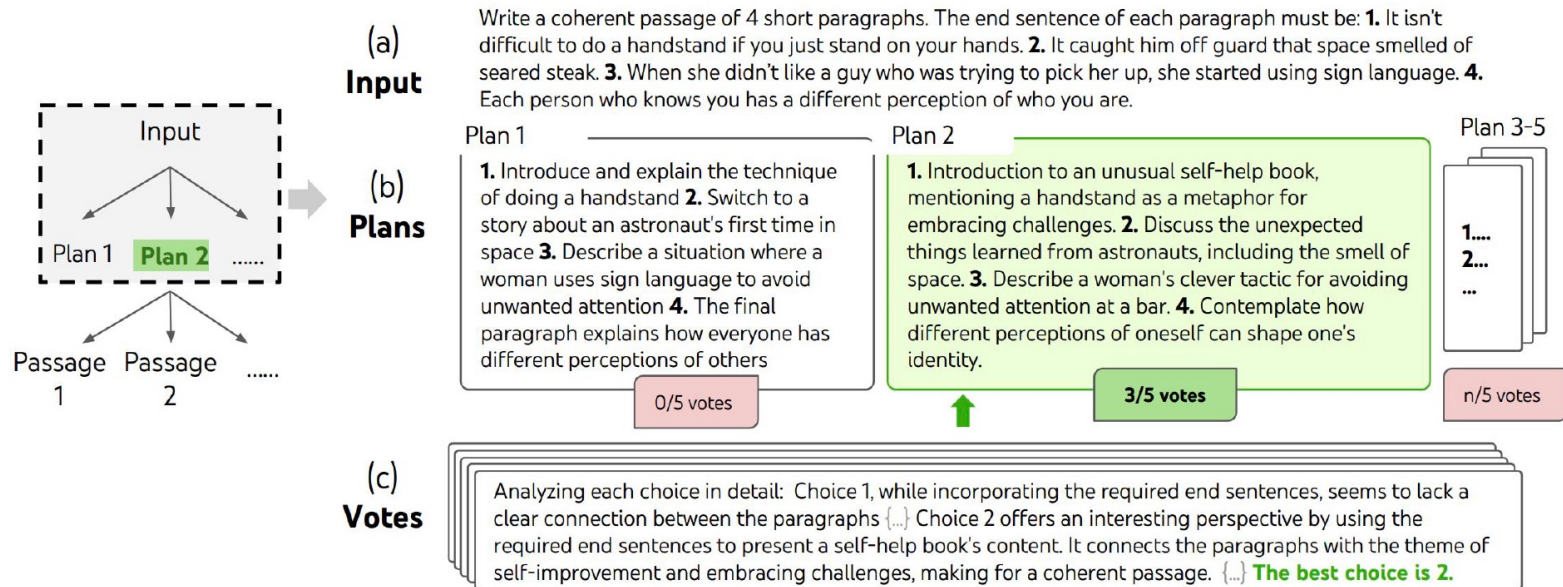
Tree-of-Thought (ToT) Example: Game of 24



At Each Step:

- ❖ Thought Generation: Prompt the LLM to propose possible next "thinking" steps
- ❖ Thought Evaluation: Prompt the LLM to evaluate how promising the current state is

Voting-based State Evaluation

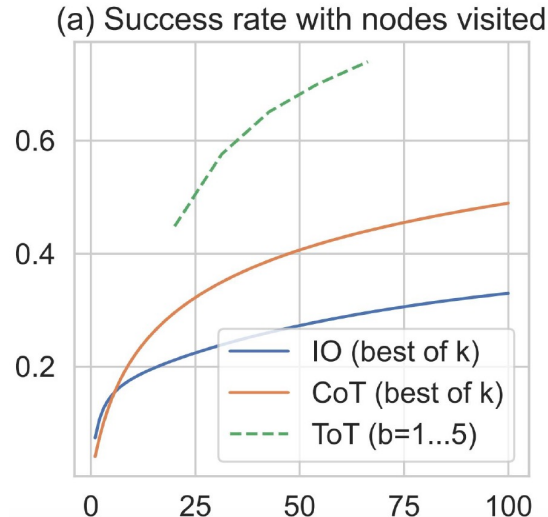


LLM selects the best state among candidates:

❖ LLM votes multiple times, then selects the majority vote as the final choice

Tree-of-Thought (ToT) Performance for Game of 24

Method	Success
IO prompt	7.3%
CoT prompt	4.0%
CoT-SC ($k=100$)	9.0%
ToT (ours) ($b=1$)	45%
ToT (ours) ($b=5$)	74%



- ❖ ToT w/ Breadth-First Search (BFS) scales better than Standard Prompting and CoT w.r.t. Token Budget
- ❖ One should try to integrate more advanced search algorithms, e.g. Monte-Carlo Tree Search (MCTS)
 - But need a good LLM + Prompt design for Self-evaluation !

Summary of Type 2 approaches (expanding Width of Exploration)

- ❖ We can further scale the inference-time compute by sampling multiple branches in the solution space
- ❖ Consistency-based selection is simple, effective and generally applicable
 - Self-Consistency: Marginalize out Reasoning Paths and Select based on the final answer
 - Code generation: Reranking based on Execution Consistency
- ❖ When LLM Self-evaluation works well: Search in the Partial Solution space can help !

Type 3 Approaches:

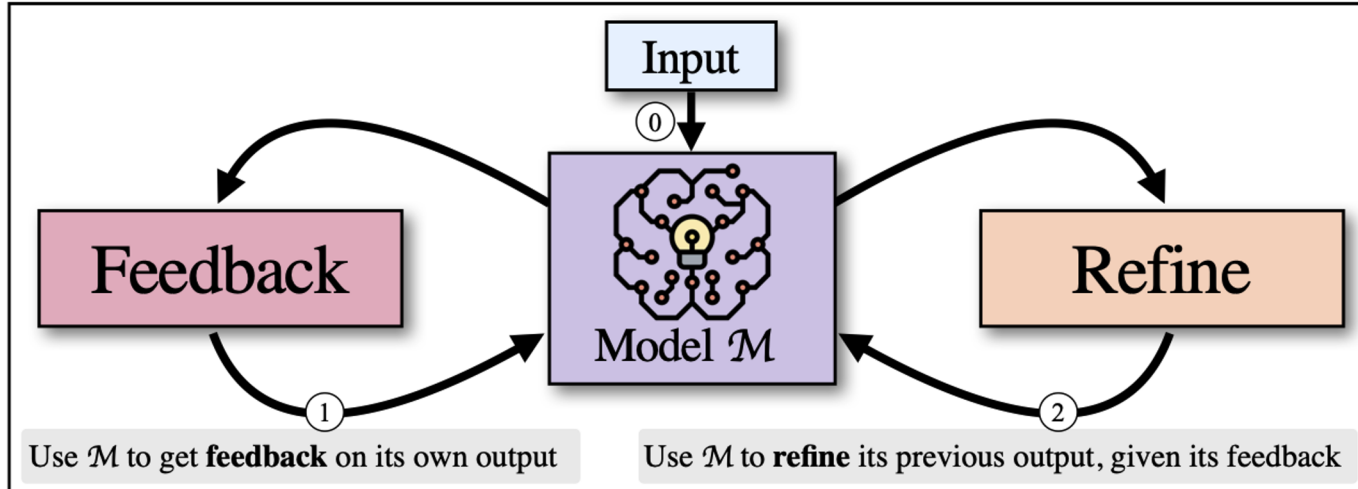
Iterative Self-improvement by increasing the Depth of Exploration Path(s)
before reaching the final solution

Motivation for increasing the LLM's Depth of Exploration Path

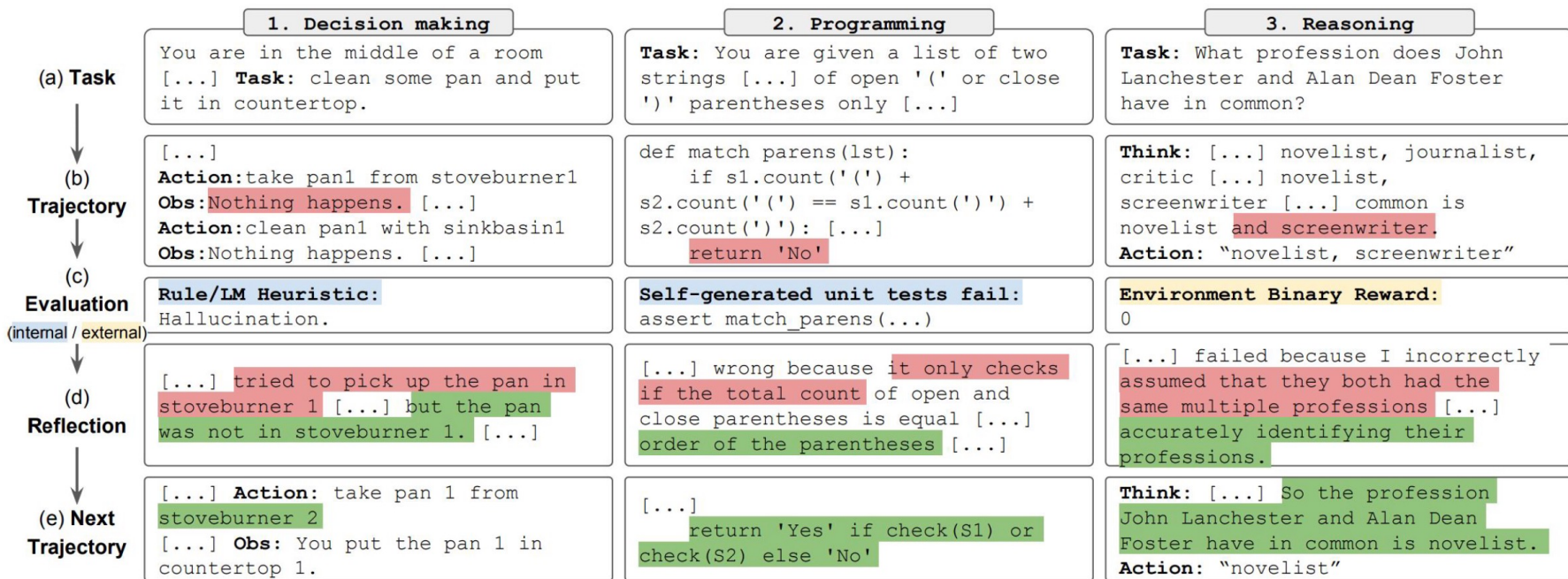
- ❖ Even Humans often make (sometimes trivial) mistakes at first thought
 - ❖ Although sampling multiple solutions (i.e. increase the width of exploration space) can reduce mistakes by considering more than one single prediction, it is still suboptimal because
 - NO feedback loop to correct the mistakes after a complete solution is generated
- In contrast,
- under the Inference-time self-improvement approach, LLM iteratively improves its own response for the given task.

Self-Refine

Basic idea: prompt an LM to critique/revise its own outputs iteratively.



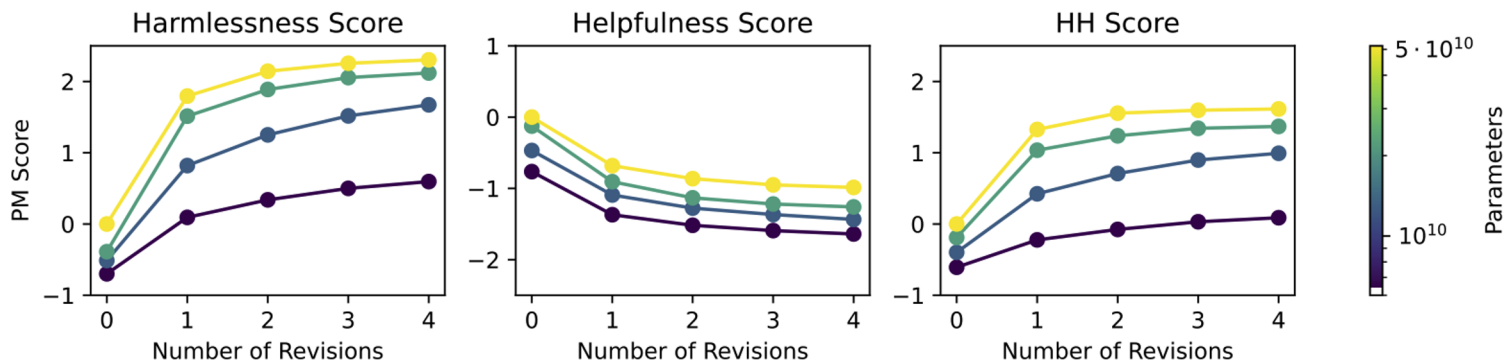
Reflection (Reflexion) and Self-Refine



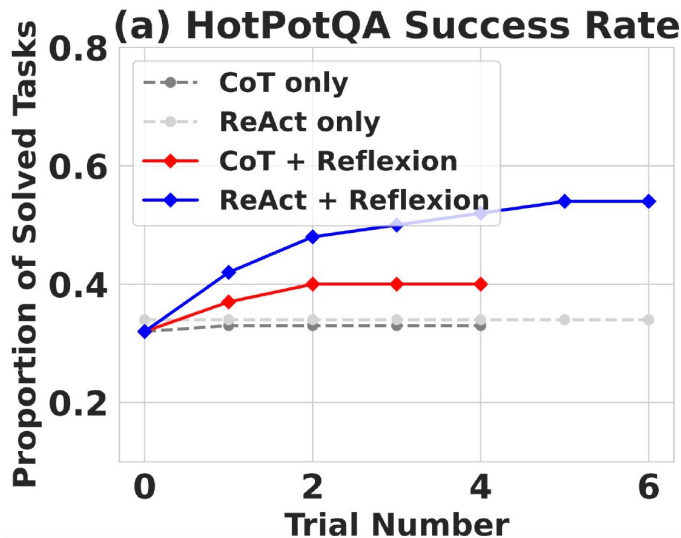
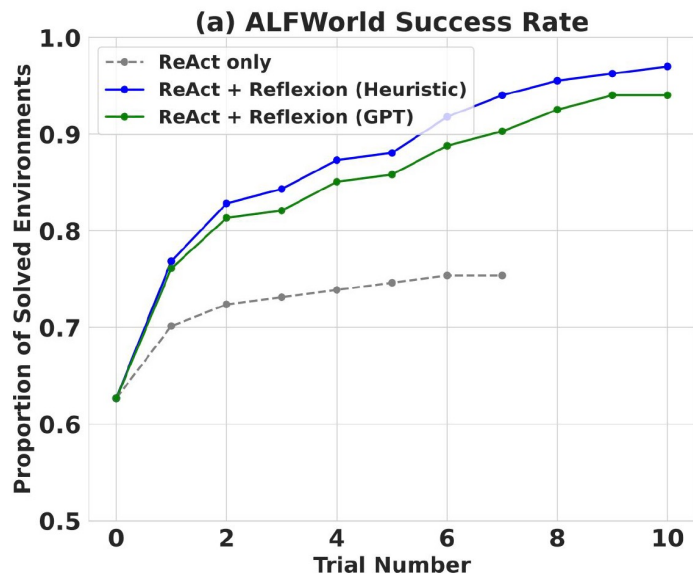
- ❖ LLM generates feedback on its output. Use external evaluation (oracle) when available.
- ❖ LLM self-refines its output given both internal feedback and external evaluation.

Initial Findings regarding Performance of Self-Refine

Seems like it works reasonably well on easy tasks like chatbot harmlessness or summarization.



Self-reflection and Self-refinement work **ONLY WHEN** Good External Evaluation (Oracle) is available



- ❖ Reflexion improves on tasks with **effective** evaluation heuristics, e.g., ALFWorld.
- ❖ On HotPotQA: **the external evaluation gives the answer correctness** at each reflection step.

Self-correction without Oracle feedback hurts Reasoning Performance

		GSM8K	CommonSenseQA	HotpotQA
GPT-3.5	Standard Prompting	75.9	75.8	26.0
	Self-Correct (Oracle)	84.3	89.7	29.0
GPT-4	Standard Prompting	95.5	82.0	49.0
	Self-Correct (Oracle)	97.5	85.5	59.0

		# calls	GSM8K	CommonSenseQA	HotpotQA
GPT-3.5	Standard Prompting	1	75.9	75.8	26.0
	Self-Correct (round 1)	3	75.1	38.1	25.0
	Self-Correct (round 2)	5	74.7	41.8	25.0
GPT-4	Standard Prompting	1	95.5	82.0	49.0
	Self-Correct (round 1)	3	91.5	79.5	49.0
	Self-Correct (round 2)	5	89.0	80.0	43.0

- ❖ Oracle: utilize the ground truth answer for correction
- ❖ Without oracle feedback, LLMs need to judge the response correctness themselves
- ❖ LLMs can wrongly judge the correctness of its predictions, leading to worse performance after self-correction

General-Purpose Feedback Prompt variants **DO NOT** improve Performance

	# calls	GSM8K	CommonSenseQA
Standard Prompting	1	91.5	84.0
<i>Feedback Prompt:</i> Assume that this answer could be either correct or incorrect. Review the answer carefully and report any serious problems you find.			
Self-Correct (round 1)	3	88.0	81.5
Self-Correct (round 2)	5	90.0	83.0
<i>Feedback Prompt:</i> Review your previous answer and determine whether it's correct. If wrong, find the problems with your answer.			
Self-Correct (round 1)	3	90.0	74.5
Self-Correct (round 2)	5	90.0	81.0
<i>Feedback Prompt:</i> Verify whether your answer is correct, and provide an explanation.			
Self-Correct (round 1)	3	91.0	81.5
Self-Correct (round 2)	5	91.0	83.5

- ❖ Edit the feedback prompt affects the self-correction behavior (tendency to keep the initial response), but none of them significantly improves over the initial performance.

Luckily, some applications do have an Oracle available and can be leveraged !

Self-debug naturally leverage unit tests as oracle

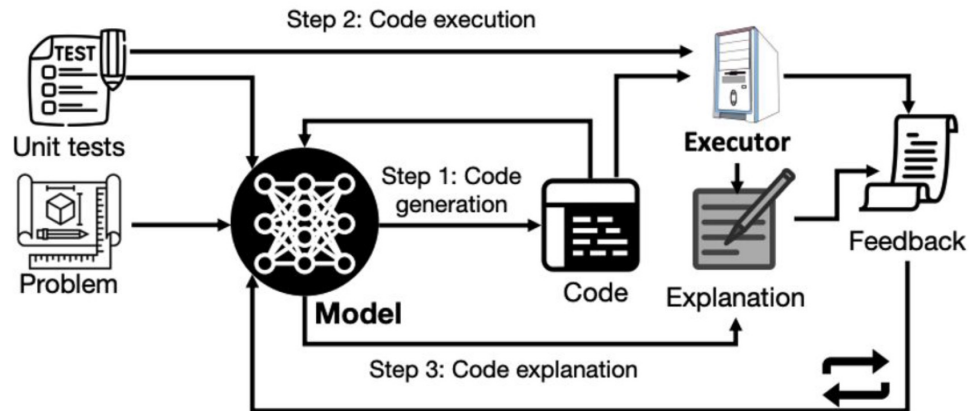


Figure 1: SELF-DEBUGGING for iterative debugging using a large language model. At each debugging step, the model first generates new code, then the code is executed and the model explains the code. The code explanation along with the execution results constitute the feedback message, which is then sent back to the model to perform more debugging steps. When unit tests are not available, the feedback can be purely based on code explanation.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, Denny Zhou. [Teaching Large Language Models to Self-Debug](#). arXiv:2302.00093 [cs.CL], 2023. ICLR 2024.

❖ **Code execution provides natural external feedback**: humans often debug better within an IDE !

Self-Debugging with different Feedback Formats

```
[C++] =  
int count_trailing_zeroes_factorial_number ( int  
n ) {  
    int count = 0;  
    for ( int i = 5; n / i >= 1; i *= 5 )  
        count += n / i;  
    return count;  
}
```

```
[Original Python] =  
def count_trailing_zeroes_factorial_number(n):  
    cnt = 0  
    i = 5  
    while ((n / i) >= 1):  
        cnt = (cnt + (n / i))  
        i *= 5  
    return cnt
```

[Simple Feedback] =
The above Python translation does not do the same thing as the C++ code. Correct the Python translation.

[UT Feedback] =
The Python translation does not do the same thing as the C++ code. These are the results of failed unit tests that test whether the Python translation's outputs match the C++ program's outputs:

Failed: assert
count_trailing_zeroes_factorial_number(9) == 1
Actual Result: 1.8

Correct the translation.

```
[Revised Python #n] =  
def count_trailing_zeroes_factorial_number(n):  
    cnt = 0  
    i = 5  
    while ((n / i) >= 1):  
        cnt = (cnt + (n // i))  
        i *= 5  
    return cnt
```

[C++ Explanation] =
The code is an implementation of counting the number of trailing zeroes in the factorial of a given number.
The variable count is used to record the number of trailing zeroes.
for (int i = 5; n / i >= 1; i *= 5)
The loop iterates until the quotient of the given number and i (the divisor) is greater than or equal to 1.
count += n / i;
When the loop is executed, the quotient of the given number and i will be added to the variable count.
return count;
Finally, the function returns the number of trailing zeroes.

[Python Explanation] =
The code is an implementation of counting the number of trailing zeroes in the factorial of a given number.
cnt = 0;
The variable cnt is used to record the number of trailing zeroes.
i = 5
The variable i is used as the divisor.
while ((n / i) >= 1):
The loop iterates until the quotient of the given number and i (the divisor) is greater than or equal to 1.
cnt = (cnt + (n / i))
When the loop is executed, the quotient of the given number and i will be added to the variable cnt.
i *= 5
The variable i will be multiplied by 5.
return cnt
Finally, the function returns the number of trailing zeroes.

[Trace] =
1. Initialize 'n' as 9, 'cnt' as 0, 'i' as 5.
2. Enter the while loop since '(9 / 5)' is greater or equal to 1.
3. 'cnt' is assigned 'cnt + (9 / 5)' which evaluates to 1.8.
4. 'i' is multiplied by 5, becomes 25.
5. Check the condition again '(9 / 25)' is not greater or equal to 1.
Exit the while loop.
6. Return 'cnt' which is 1.8.

The incorrect translation fails because the division 'n / i' in Python returns a float value, whereas in C++ it returns an integer value. This issue affects the value assigned to 'cnt' and results in an incorrect output.

- ❖ **Simple**: a short universal feedback for all wrong code
- ❖ **Unit test feedback**: include the execution results
- ❖ **Code explanation**: line-by-line explanation of the implementation
- ❖ **Trace**: line-by-line simulation of the execution trace

Self-Debugging Performance

(a) Results on the Spider development set.

Spider	Codex	GPT-3.5	GPT-4	StarCoder
Baseline	81.3	71.1	73.2	64.7
Simple	81.3	72.2	73.4	64.9
+Expl.	84.1	72.2	73.6	64.9

(b) Results on TransCoder.

TransCoder	Codex	GPT-3.5	GPT-4	StarCoder
Baseline	80.4	89.1	77.3	70.0
Simple	89.3	91.6	80.9	72.9
UT	91.6	92.7	88.8	76.4
+ Expl.	92.5	92.7	90.4	76.6
+ Trace.	87.9	92.3	89.5	73.6

(c) Results on MBPP.

MBPP	Codex	GPT-3.5	GPT-4	StarCoder
Baseline	61.4	67.6	72.8	47.2
Simple	68.2	70.8	78.8	50.6
UT	69.4	72.2	80.6	52.2
+ Expl.	69.8	74.2	80.4	52.2
+ Trace.	70.8	72.8	80.2	53.2

- ❖ Self-debugging consistently boosts the performance across different LLMs
- ❖ More informative feedback further improves the debugging performance

Multi-agent Debate

Basic idea: prompt multiple LM instances to debate each other to find the answer.

Question: Carla is downloading a 200 GB file. Normally she can download 2 GB/minute, but 40% of the way through the download, Windows forces a restart to install updates, which takes 20 minutes. Then Carla has to restart the download from the beginning. How long does it take to download the file?

Round 1

ChatGPT: Before the restart, Carla was able to download 40% of the file, which is $0.4 * 200 \text{ GB} = 80 \text{ GB}$. At a rate of 2 GB/minute, it would take her: $\text{time} = 80 \text{ GB} \div 2 \text{ GB/minute} = 40 \text{ minutes}$.

Because of the restart, Carla had to start over. This means she had to download the entire 200 GB file. The time it takes to download the remaining 120 GB is: $\text{time} = 120 \text{ GB} \div 2 \text{ GB/minute} = 60 \text{ minutes}$.

But Carla also had to wait 20 minutes for the Windows update to install, so the total time it took to download the file is: $\text{total time} = 40 \text{ minutes} + 60 \text{ minutes} + 20 \text{ minutes} = 120 \text{ minutes}$. ❌

Bard: Carla is downloading a 200 GB file at a speed of 2 GB/minute. This means that it will take her $200 / 2 = 100 \text{ minutes}$ to download the file.

However, Windows forces a restart to install updates, which takes 20 minutes. This means that Carla will have to restart the download from the beginning. Since Carla has already downloaded 40% of the file, she only needs to download $200 - 80 = 120 \text{ GB}$. At a speed of 2 GB/minute, it will take Carla $120 / 2 = 60 \text{ minutes}$ to download the file.

Adding the time for the Windows restart, it will take Carla $80 + 20 = 100 \text{ minutes}$ to download the file. ❌

Round 2

ChatGPT : Based on the calculations provided by the other agent, the correct answer is: Before the restart, Carla was able to download 40% of the file, which is $0.4 * 200 \text{ GB} = 80 \text{ GB}$. At a rate of 2 GB/minute, it would take her: $\text{time} = 80 \text{ GB} \div 2 \text{ GB/minute} = 40 \text{ minutes}$

However, because of the restart, Carla had to start over. This means she had to download the entire 200 GB file. The time it takes to download the entire 200 GB file is: $\text{time} = 200 \text{ GB} \div 2 \text{ GB/minute} = 100 \text{ minutes}$

And since Carla had to wait 20 minutes for the Windows update to install, the total time it took to download the file is: $\text{total time} = 40 \text{ minutes} + 100 \text{ minutes} + 20 \text{ minutes} = 160 \text{ minutes}$. ✅

BUT Multi-agent (LLM) debate performs worse than Self-consistency

	# responses	GSM8K
Standard Prompting	1	76.7
Self-Consistency	3	82.5
Multi-Agent Debate (round 1)	6	83.2
Self-Consistency	6	85.3
Multi-Agent Debate (round 2)	9	83.0
Self-Consistency	9	88.2

- ❖ **Multi-agent debate**: prompt the LLM to review multiple responses and give an updated one.

Vs.

- ❖ **Self-consistency** selects the response with the most common final answer.
- ❖ With the right prompt, Multi-agent does not outperform SC
- ❖ Without a good evaluator, multi-agent debate does not effectively utilize the token budget.

Trade-off Analysis of Inference-time (aka Test-time) Scaling:

How to best Utilize the Token Budget for the LLM ?

Type 1 vs Type 2 vs Type 3 approaches to make LLM reason ?

Clear Evidence of Inference-Time (Test-Time) Scaling

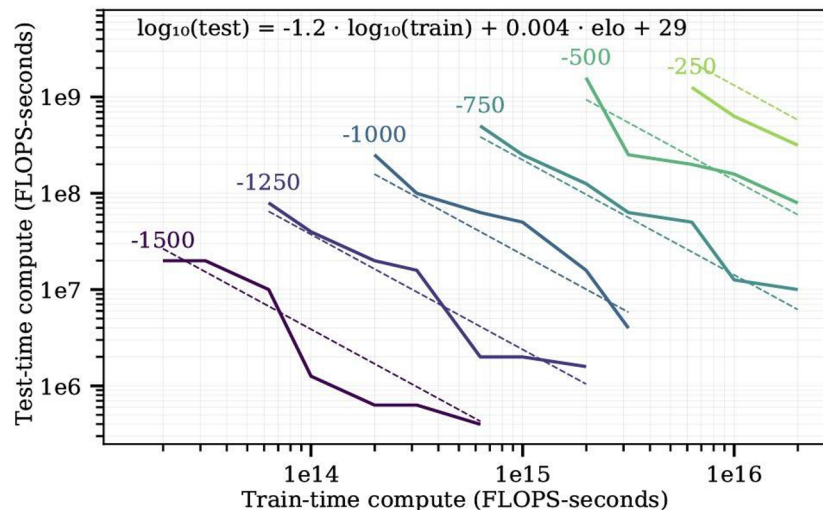
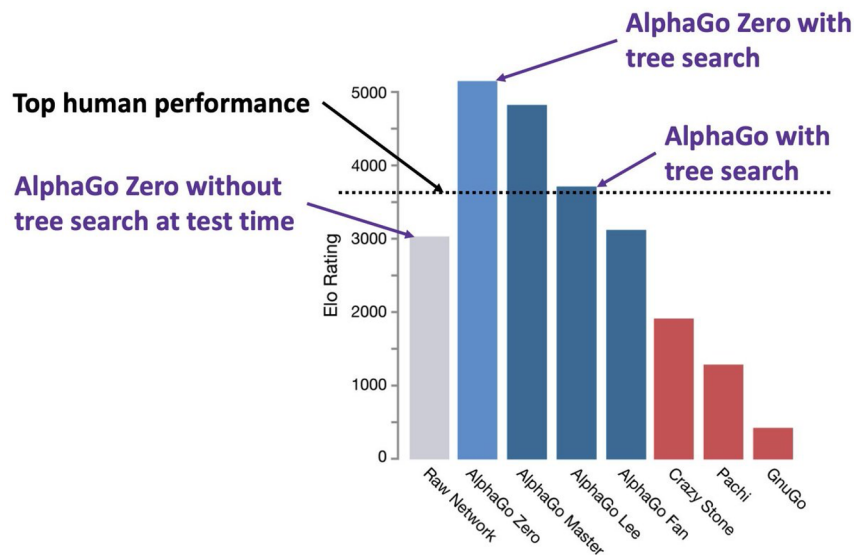


Fig. 9. The trade-off between train-time compute and test-time compute. Each dotted line gives the minimum train-test compute required for a certain Elo on a 9×9 board

- ❖ Giving AI additional test-time compute can greatly improve performance.
- ❖ However, previous demonstrations are mainly limited to specific tasks.

How to best Utilize the Token Budget in General ?

- ❖ How to balance inference-time budget for generating multiple samples (solutions and/or partial solutions) ?
 - ❖ Generating in Parallel vs. Sequential ?
- ❖ In general, different problems may benefit from different inference-time compute strategies
 - ❖ One could obtain N answers and find the most common one (i.e. Self-Consistency, or Universal Self-Consistency)
 - ❖ One could do Best-of-N with a Learned Verifier
 - ❖ One could use the LLM to revise and correct it's own responses iteratively
 - ❖ ...

All these techniques induce the use of more token budget to generate the final solution!

Experimental Setup

Q: Given a challenging input query, how can we enable language models to most effectively make use of additional computation at inference (test) time so as to improve the accuracy of their response?

There are many different ways we could utilize test-time compute.

- We could generate N solutions and then apply Self-Consistency / USC criteria for selection.
- We could do best of N with a learned verifier.
- We could use the model to revise and correct it's own responses iteratively.
- etc...

Different problems may benefit from different test-time compute strategies.

Unifying Perspective on Test-time Computation: Proposer and Verifier

- Try to scale test-time compute via two mechanisms: modifying the LM's inputs or its outputs.
 - *Input level: modify the prompt.*
 - ❖ Improve the proposal distribution from which outputs are sampled.
e.g. train the model to revise proposals rather than sampling them all in parallel.
 - *Output level: guide outputs in some way. For examples:*
 - ❖ Search against intermediate steps in a verifier.
 - ❖ Sample N outputs in parallel, N revisions, or some balance of these two.
 - Sampling in parallel may act as a global search for the right high-level approach.
 - Better for hard problems.
 - Sampling revisions might be more useful if the answer is already on the right track.
 - Better for easy problems.

=> Different problems may benefit from different test-time compute strategies.

Compute Optimal Scaling of Inference-time Compute

Select the best possible test-time compute hyper-params for a given question and test-time budget.

$$\theta_{q,a^*(q)}^*(N) = \operatorname{argmax}_{\theta} \left(\mathbb{E}_{y \sim \text{Target}(\theta, N, q)} \left[\mathbb{1}_{y=y^*(q)} \right] \right),$$

Hyper-parameters could be:

- What search algorithm to use (e.g. best-of-N verses beam search).
- Whether to sample N answers in parallel or N revisions in sequence.

Question Difficulty for Compute Optimal Scaling

Use question difficulty as a sufficient statistic for practically estimating compute optimal scaling.

- ❖ For a given difficulty level, select the best performing hyperparameters.

Oracle difficulty

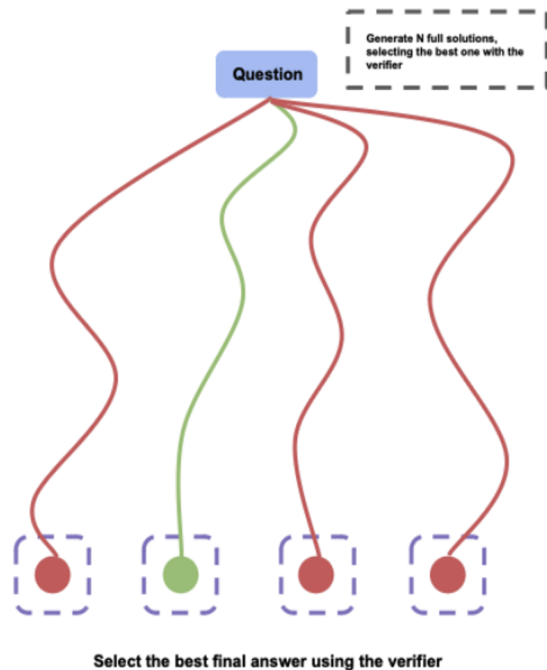
- Sample 2048 outputs per question and bin questions into 5 quantiles by how many of the samples were correct.

Predicted difficulty

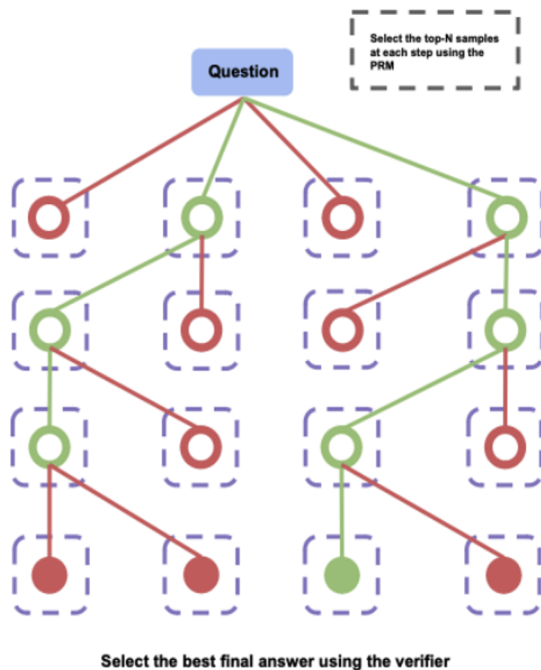
- Do the same thing but use a verifier's predicted correctness score instead of ground truth correctness information.

Scaling Inference-Time Compute with Verifiers

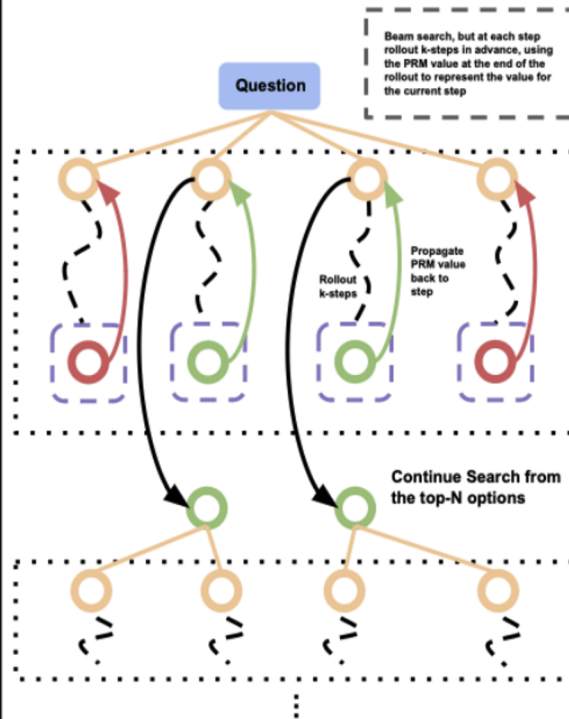
Best-of-N



Beam Search



Lookahead Search



Key:



= Apply Verifier



= Full Solution



= Intermediate solution step



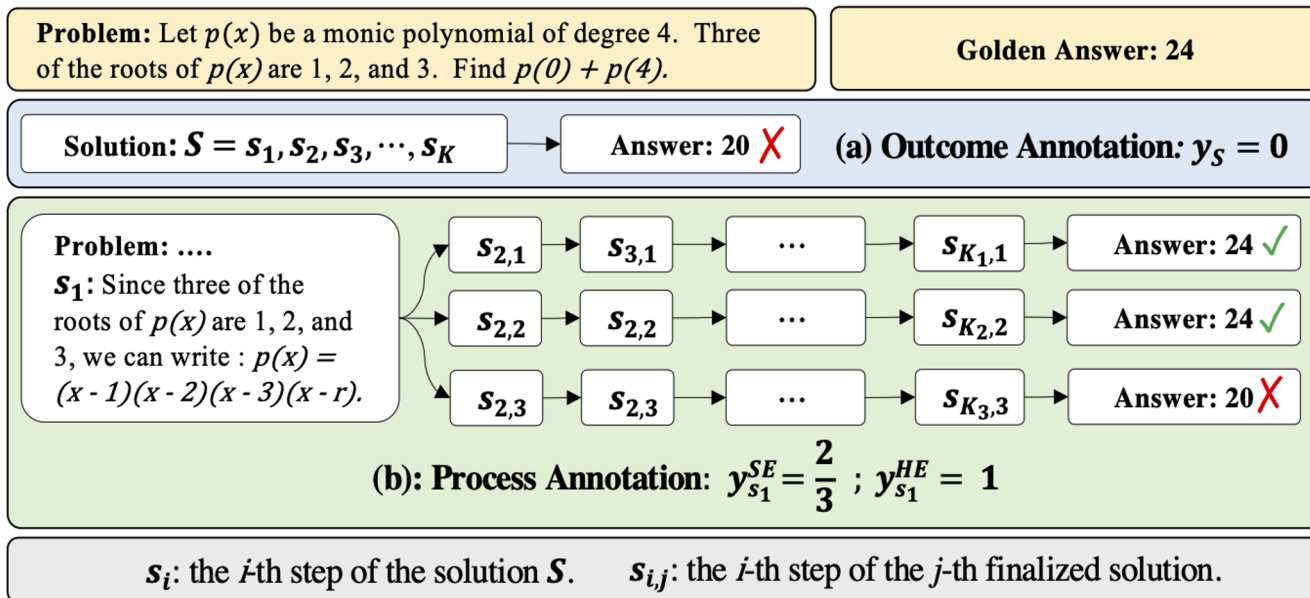
= Selected by verifier



= Rejected by verifier

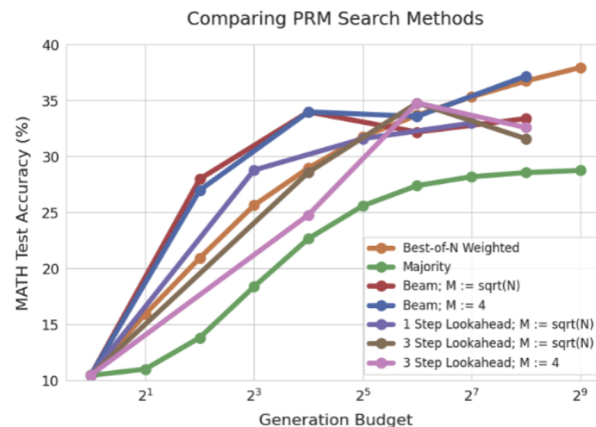
Training a Process Based Verifier

- For search, the verifier should score each step in a solution.
 - Prior work [lightman et al. 2023] used human labels of per-step correctness for supervision.
 - This work instead follows the approach from MathShepard [wang et al. 2024] to learn a value function.



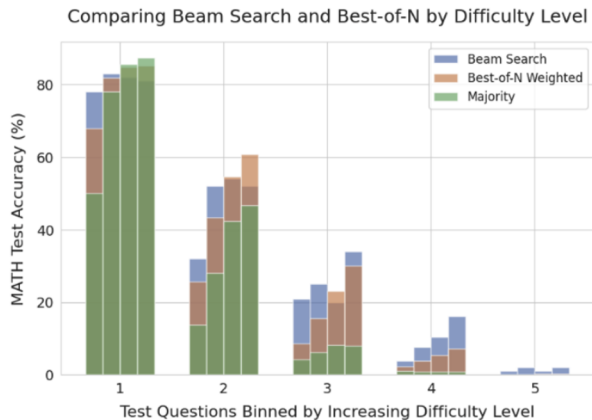
Comparing Search Methods

- Beam search performs best at small budgets.
- At large budgets, Best-of-N performs similarly to beam search.
- Lookahead generally underperforms due to the high cost of the lookahead rollouts.
- All verifier search methods outperform the majority baseline.

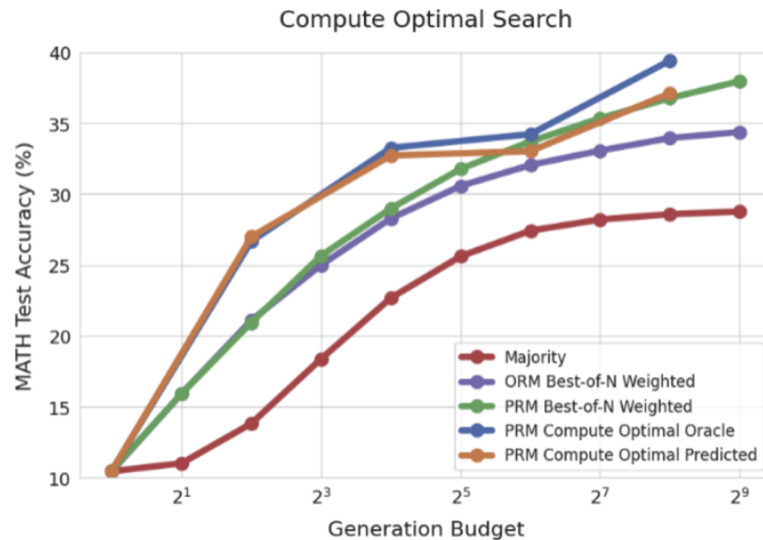


Performance Breakdown by Question Difficulty

- On easy questions, beam-search shows some signs of over-optimization.
- On medium/hard questions, beam-search outperforms best-of-N.
- On the hardest questions, all methods struggle.



Compute Optimal Search



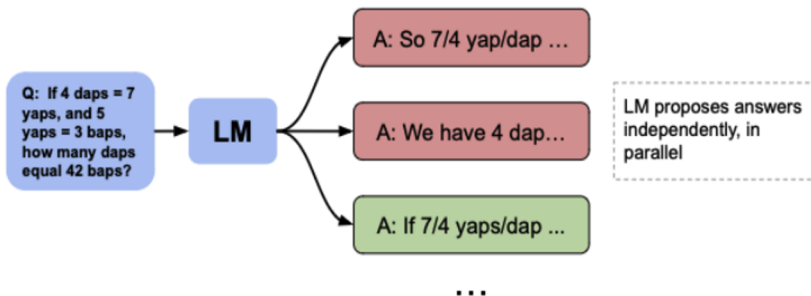
By selecting the best performing search algorithm at each difficulty level, we can nearly outperform best-of-N using up to **4x** less test-time compute.

Takeaways for Scaling Inference-time Compute With Verifiers

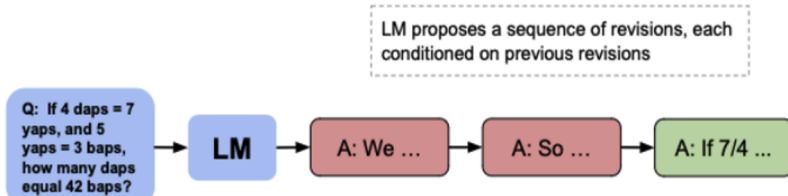
- The efficacy of a search method depends on the budget and the question.
- Beam search is more effective on harder questions and at lower budgets.
- Best-of-N is more effective on easier questions and at higher budgets.
- By selecting the best setting for each question, we can nearly outperform best-of-N using up to **4x** less test-time compute.

Scaling Inference-time Compute With Revisions

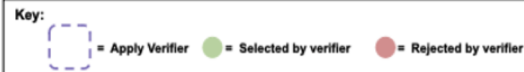
Parallel Sampling



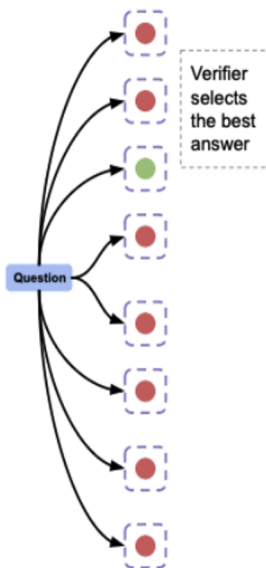
Sequential Revisions



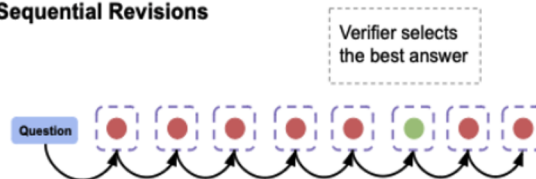
Using Revision Model + Verifier at Inference Time



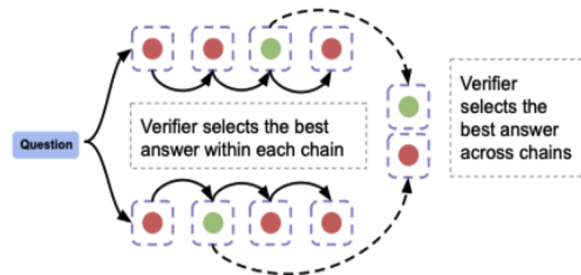
Parallel Best-of-N



Sequential Revisions



Combining Sequential / Parallel

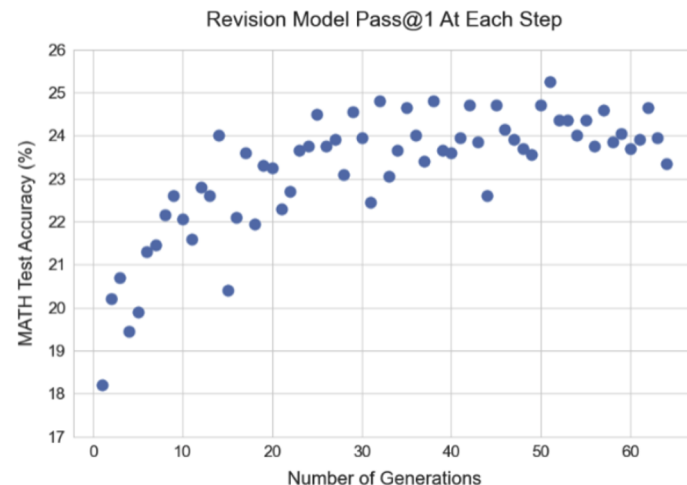


Fine-tuning a Revision Model

Fine tune a model to iteratively revise answers in context.

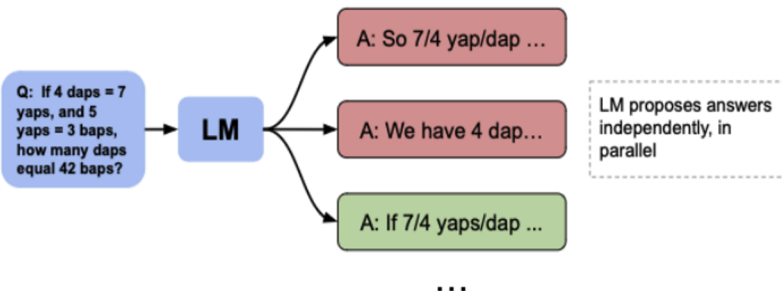
Procedure:

1. Sample N solutions to a question from the base LM.
2. Create a chain of incorrect answers followed by a correct answer.
3. Finetune the model to generate the correct answer conditioned on the chain.

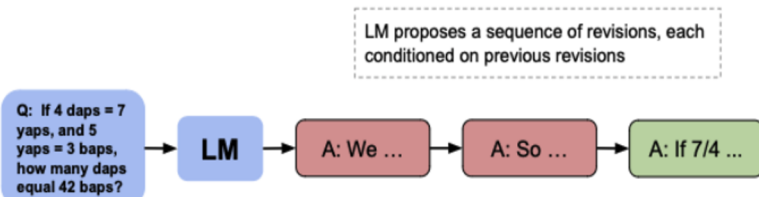


Using a Verifier with the Revision Model

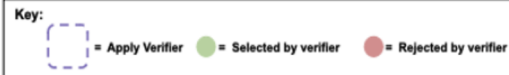
Parallel Sampling



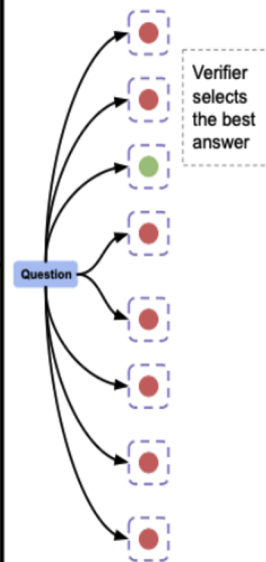
Sequential Revisions



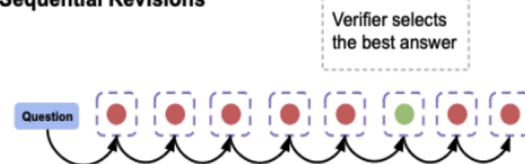
Using Revision Model + Verifier at Inference Time



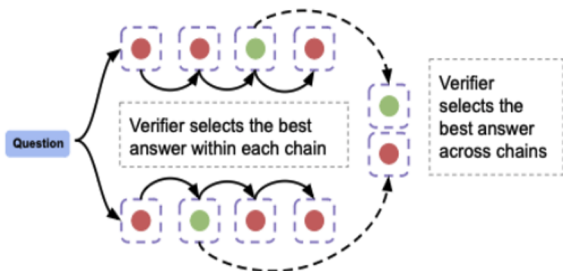
Parallel Best-of-N



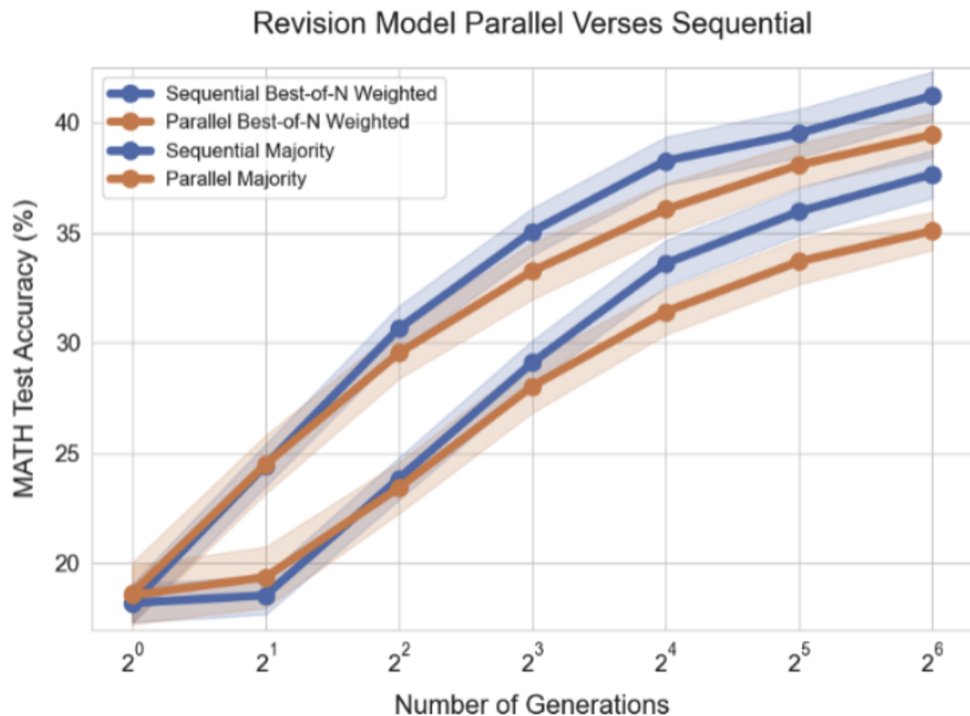
Sequential Revisions



Combining Sequential / Parallel

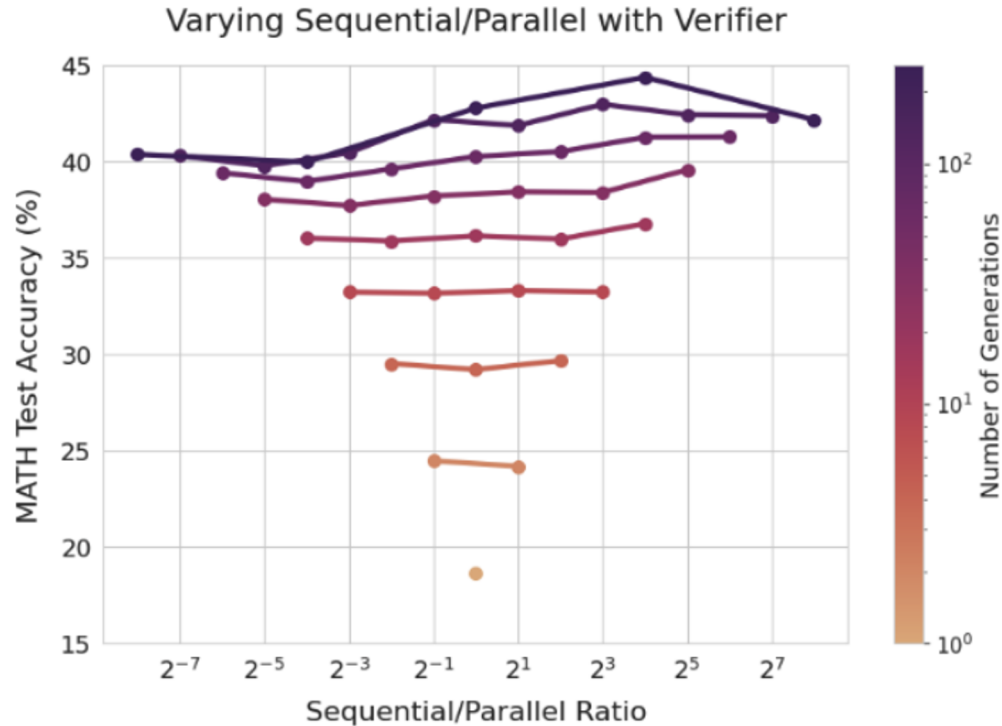


Comparing Sequential and Parallel Sampling



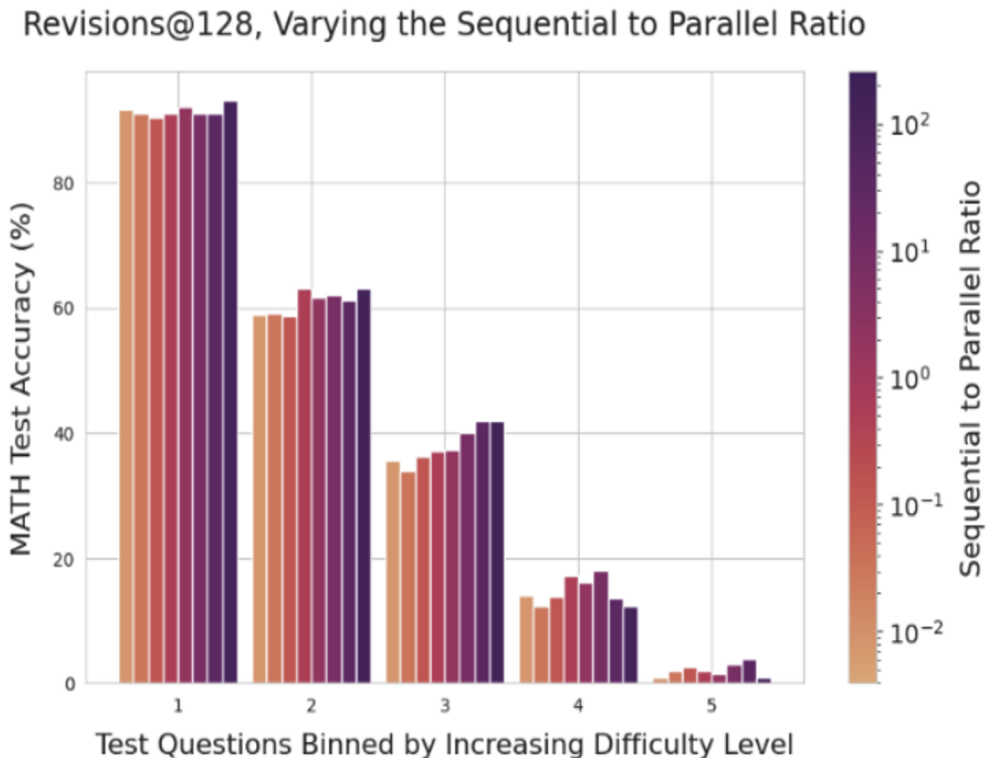
Sampling N outputs in sequence from the model outperforms sampling N in parallel.

Comparing Sequential and Parallel Sampling



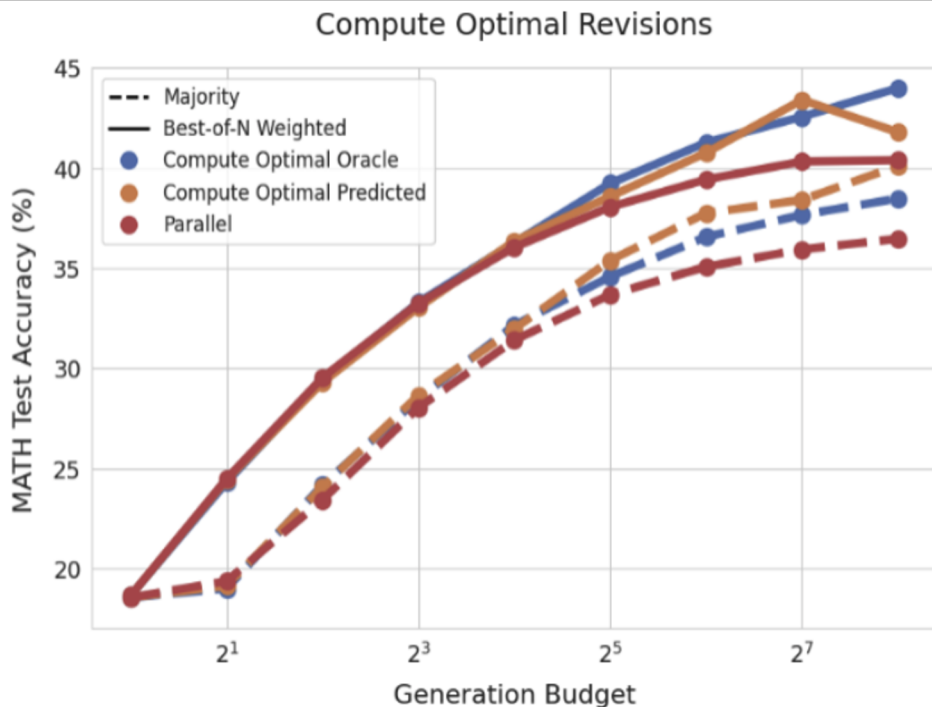
In some cases there is an ideal ratio of sequential to parallel test-time compute.

Comparing Sequential and Parallel Sampling



This ideal ratio also depends on the difficulty of the question at hand.

Scaling Inference-time Compute With Revisions



- ❖ By selecting the best performing ratio at each difficulty level, we can outperform parallel sampling using up to **4x** less test-time compute.

Takeaways for Scaling Inference-time Compute With Revisions

- There exists a tradeoff between sequential (e.g. revisions) and parallel (e.g. standard best-of-N) test-time computation.
- The ideal ratio depends on the compute budget and the question at hand.
- Easier questions benefit more from sequential revisions.
- Harder questions perform best with an ideal ratio of sequential and parallel.
- By optimally selecting the ideal ratio, we can outperform parallel sampling using up to **4x** less test-time compute.

Trade-off Inference-time with Pretraining Compute

Can scaling test-time compute be more effective than scaling parameters?

Trading Test-Time Compute with Train-time Compute

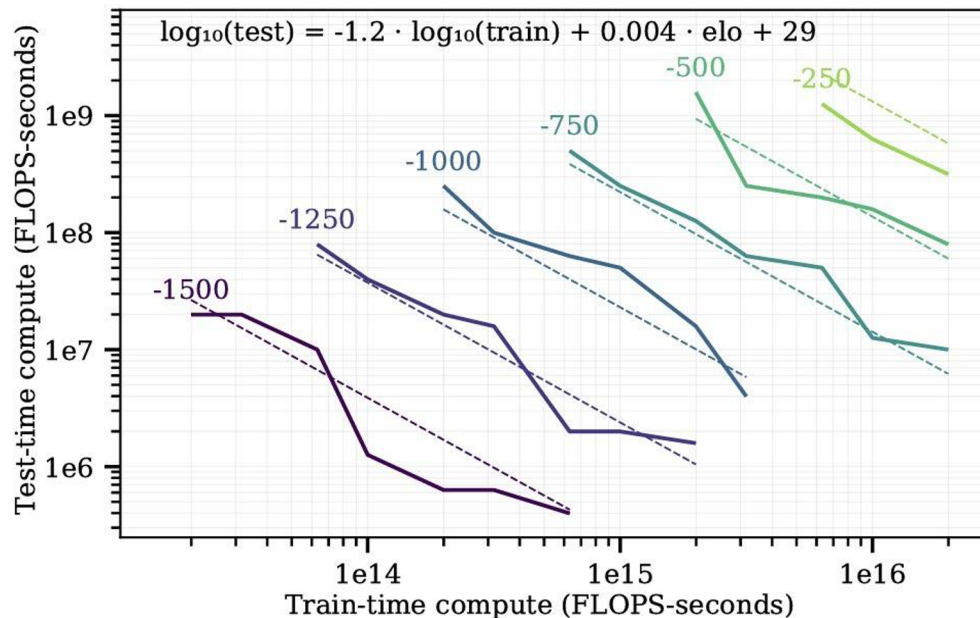


Fig. 9. The trade-off between train-time compute and test-time compute. Each dotted line gives the minimum train-test compute required for a certain Elo on a 9×9 board

- ❖ Evidence shown we can trade test-time compute with Training-time Compute
- ❖ However, previous demonstrations are mainly limited to specific tasks.

Exchanging Inference-time and Pretraining Compute

Can scaling test-time compute be more effective than scaling parameters?

- Model is pretrained with X FLOPs and we will run Y FLOPs of inference.
- If we increase the FLOPs budget by a factor of M (e.g. $\text{budget} = M(X+Y)$), should we spend it on scaling parameters or on scaling pretraining?

Exchanging Inference-time and Pretraining Compute

Can scaling test-time compute be more effective than scaling parameters?

- Model is pretrained with X FLOPs and we will run Y FLOPs of inference.
- If we increase the FLOPs budget by a factor of M (e.g. $\text{budget} = M(X+Y)$), should we spend it on scaling parameters or on scaling pretraining?
- We can either increase the parameters by a factor of M .

Exchanging Inference-time and Pretraining Compute

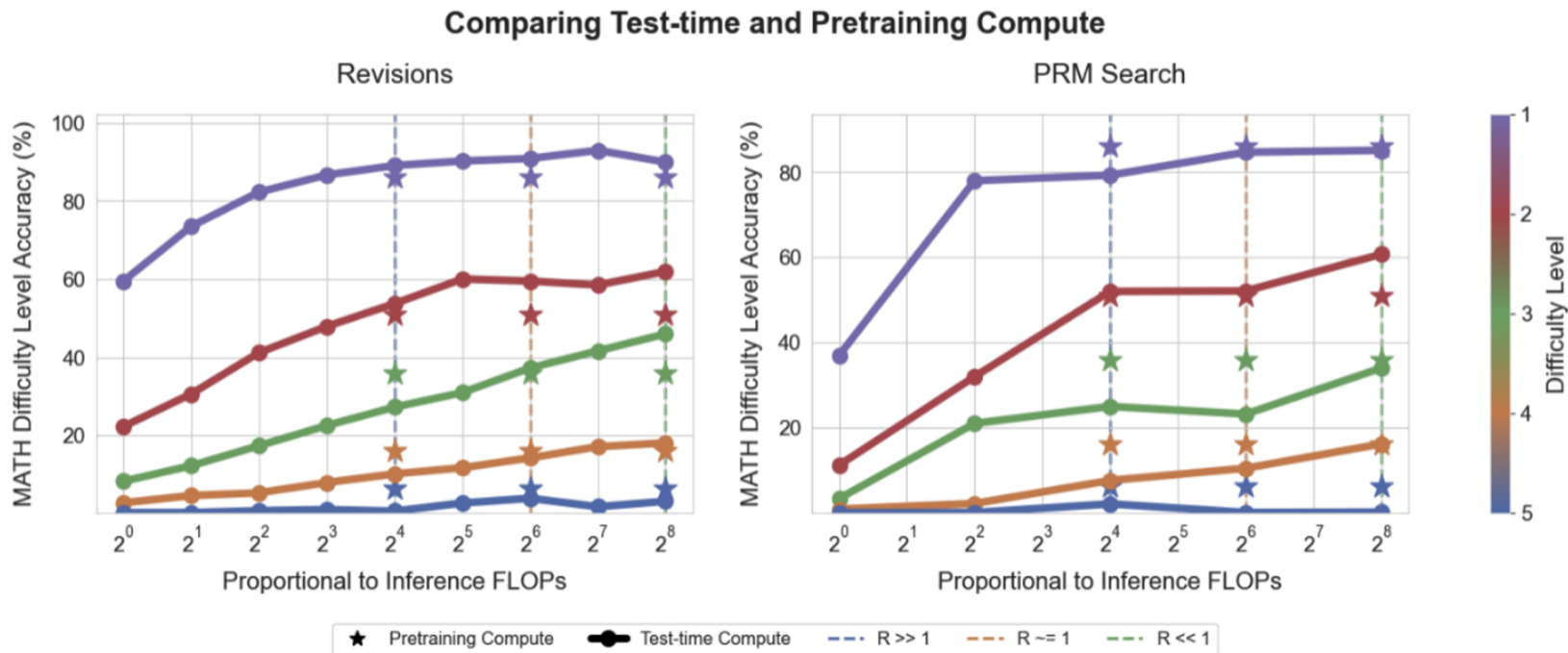
Can scaling test-time compute be more effective than scaling parameters?

- Model is pretrained with X FLOPs and we will run Y FLOPs of inference.
- If we increase the FLOPs budget by a factor of M (e.g. budget = M(X+Y)), should we spend it on scaling parameters or on scaling pretraining?
- We can either increase the parameters by a factor of M.
- Or we can keep the parameters fixed and multiply our inference budget by:

$$M + 3 \left(\frac{D_{\text{pretrain}}}{D_{\text{inference}}} \right) (M - 1).$$

This depends on a ratio of pretraining to inference tokens. We refer to the inverse of this ratio as R.

Exchanging Inference-time and Pretraining Compute



- ❖ On easy/medium difficulty questions, or in settings with low inference requirements, scaling test-time compute can be preferable to scaling parameters.

Cost Comparison:

Test-time Scaling w/ Small models Vs. Large models w/o Test-time Scaling

Model	Input Token Cost (\$ per million)	Output Token Cost (\$ per million)	AIME Score	MMLU Score	HumanEval Score	SimpleQA Accuracy	Hallucination Rate
GPT-4o	\$5	\$15	9.3%	81.5%	30.7%	38.2%	61.8%
o1 (Likely 4o size)	\$15	\$60	83%	92.3%	92.4%	47%	44%
GPT-4.5	\$75	\$150	36.7%	85.1%	38.0%	62.5%	37.1%

Takeaways / Discussion

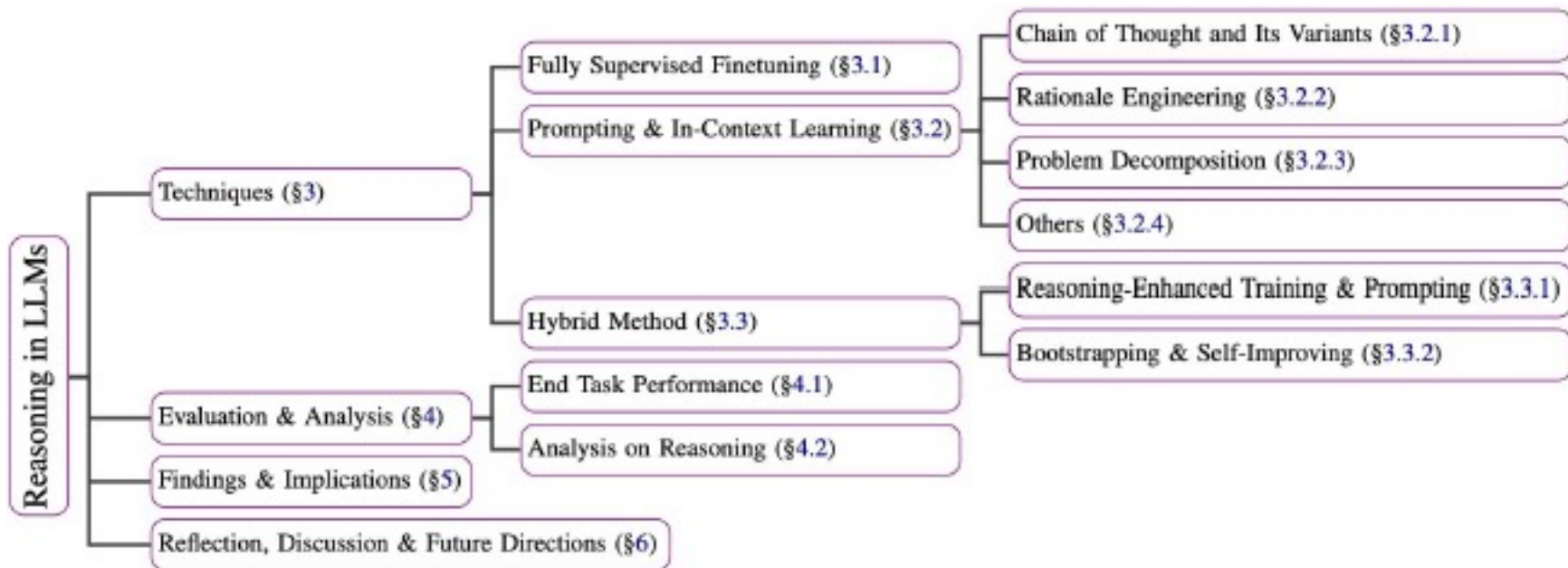
Using fairly simple methodology it is found that scaling LLM inference-time compute can greatly improve performance, and in some cases it can outperform scaling parameters.

There is much room for future work to:

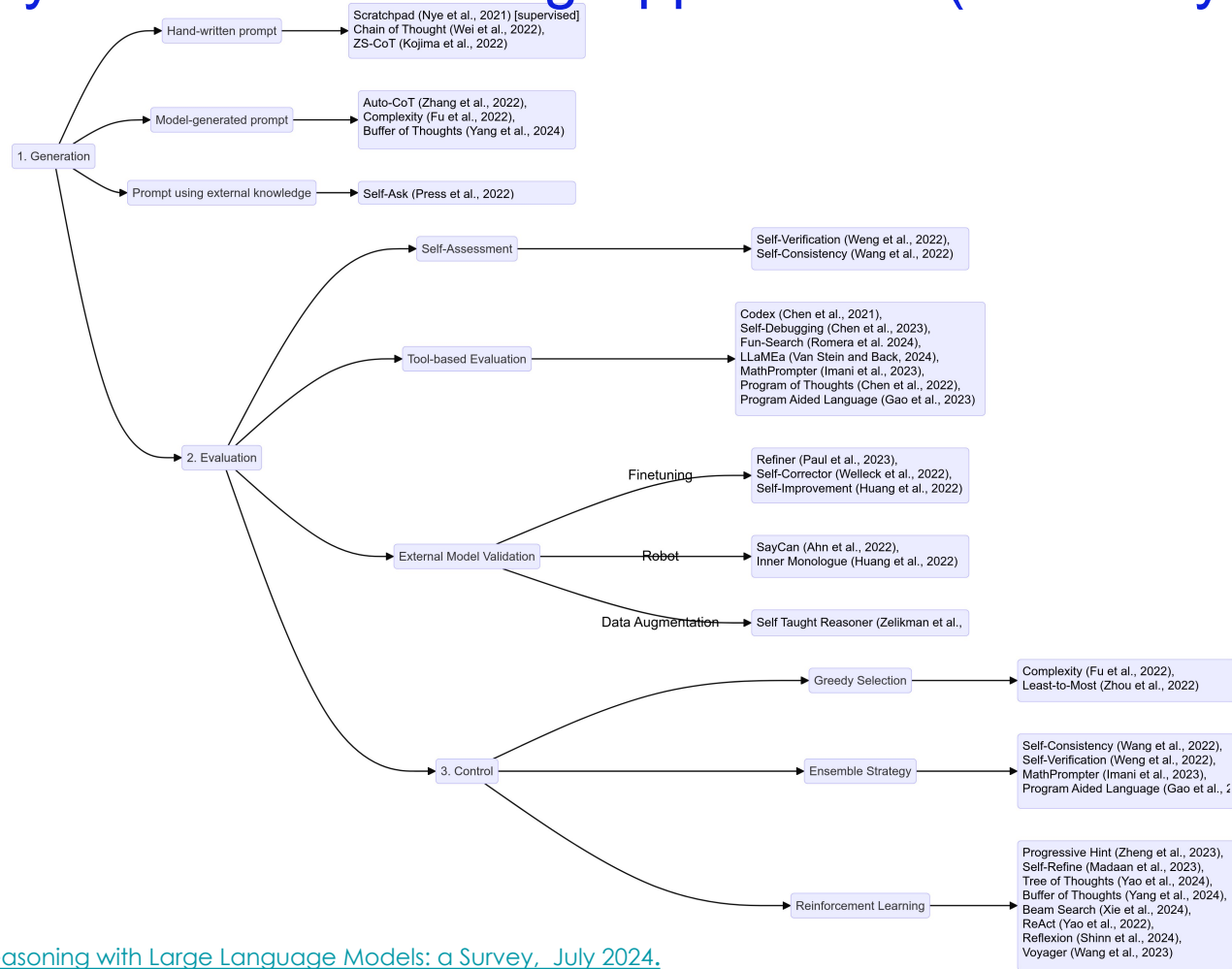
1. Improve upon the techniques and explore alternative approaches to scaling test-time compute.
2. Conduct additional analysis.

The State of LLM Reasoning Models [circa March 2025]

Recent Research on Reasoning in LLMs (circa 2023)



Taxonomy of LLM-Reasoning Approaches (circa July 2024)



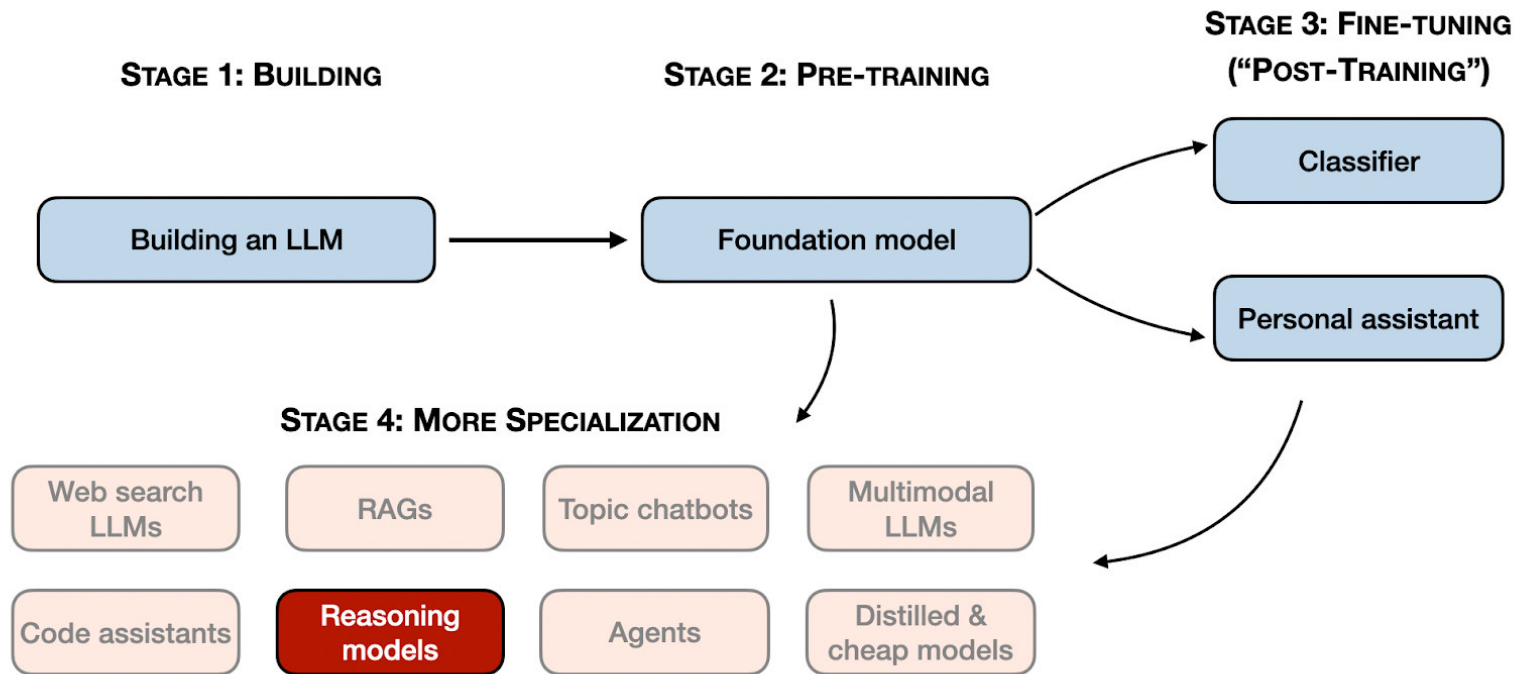
Taxonomy of LLM-Reasoning Approaches (circa July 2024)

Approach	Domain	Step generation	Step evaluation	Step control
Scratchpad [Nye et al., 2021]	math word	hand-wr/supervised	-	greedy/1prompt
Chain-of-thought [Wei et al., 2022b]	math word	hand-written	-	greedy/1prompt
ZS-CoT [Kojima et al., 2022]	math word	hand-written	-	greedy/1prompt
Auto-CoT [Zhang et al., 2022]	math word	model-generated	-	clustering
Complexity [Fu et al., 2022]	math word	hand-written	self-consistency	greedy/1prompt
Self-ask [Press et al., 2022]	math word	external knowledge	LLM	multi-hop questions
Self-verification [Weng et al., 2022]	math word	hand-written	back-verify	ensemble
Self-consistency [Wang et al., 2022b]	math word	hand-written	majority	ensemble
Codex [Chen et al., 2021]	code	-	tool-based	-
Self-debugging [Chen et al., 2023]	code	hand-written	tool-based	greedy
Fun-search [Romera-Paredes et al., 2024]	code	hand-written	tool-based	evolutionary algorithm
LLaMEa [van Stein and Bäck, 2024]	code	hand-written	tool-based	evolutionary algorithm
MathPrompter [Imani et al., 2023]	math	hand-written	tool-based	ensemble
Program-of-thoughts [Chen et al., 2022]	math word	hand-written, Codex	Python+Consist.	decouple reason/compute
Program-aided-language [Gao et al., 2023]	math word	hand-written, Codex	NLP/Python	ensemble
Refiner [Paul et al., 2023]	math word	finetune	critic model	gen/crit feedback
Self-corrector [Welleck et al., 2022]	math word	finetune	corrector model	gen/corr feedback
Self-improvement [Huang et al., 2022a]	math word	finetune	self-assessment	CoT/consistency
Say-can [Ahn et al., 2022]	robot	model-generated	external model	greedy
Inner-monologue [Huang et al., 2022b]	robot	hand-written	various	greedy
Self-taught-reasoner [Zelikman et al., 2022]	math word	finetune	augmentation	greedy/feedback
Least-to-most [Zhou et al., 2022]	math word	hand-written	self-assessment	curriculum
Progressive-hint [Zheng et al., 2023]	math word	model-generated	self-assessment	stable prompt
Self-refine [Madaan et al., 2023]	math word	model-generated	self-assessment	greedy/feedback
Tree-of-thoughts [Yao et al., 2024]	puzzles	model-generated	self-assessment	BFS/DFS
Buffer-of-thoughts [Yang et al., 2024]	math word	thought template	self-assessment	buffer manager
Beam-search [Xie et al., 2024]	math word	model-generated	self-assessment	Beam Search
ReAct [Yao et al., 2022]	action	external knowledge	self-assessment	reinforcement learning
Reflexion [Shinn et al., 2024]	decision	model-generated	ext model	reinforcement learning
Voyager [Wang et al., 2023]	Minecraft	model-generated	Minecraft	reinforcement learning

Source:

[Aske Platt et al. 2024.](#)

Reasoning as yet another “Specialized” skills for Foundation Models



Reasoning models = Models which are better in solving complex tasks such as puzzles, advanced math, coding challenges that required the introduction of more intermediate steps (aka “thinking”). By “**specialization**”, it means such skill will NOT replace other LLM applications as shown in the figure. Specialized means “Reasoning” should not be the only way LLMs use to solve problems due to **inevitable trade-offs (e.g. in resource consumption, latenc)**

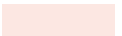
Strength and Weakness of a “Reasoning” Model and when do we need one ?

Good at

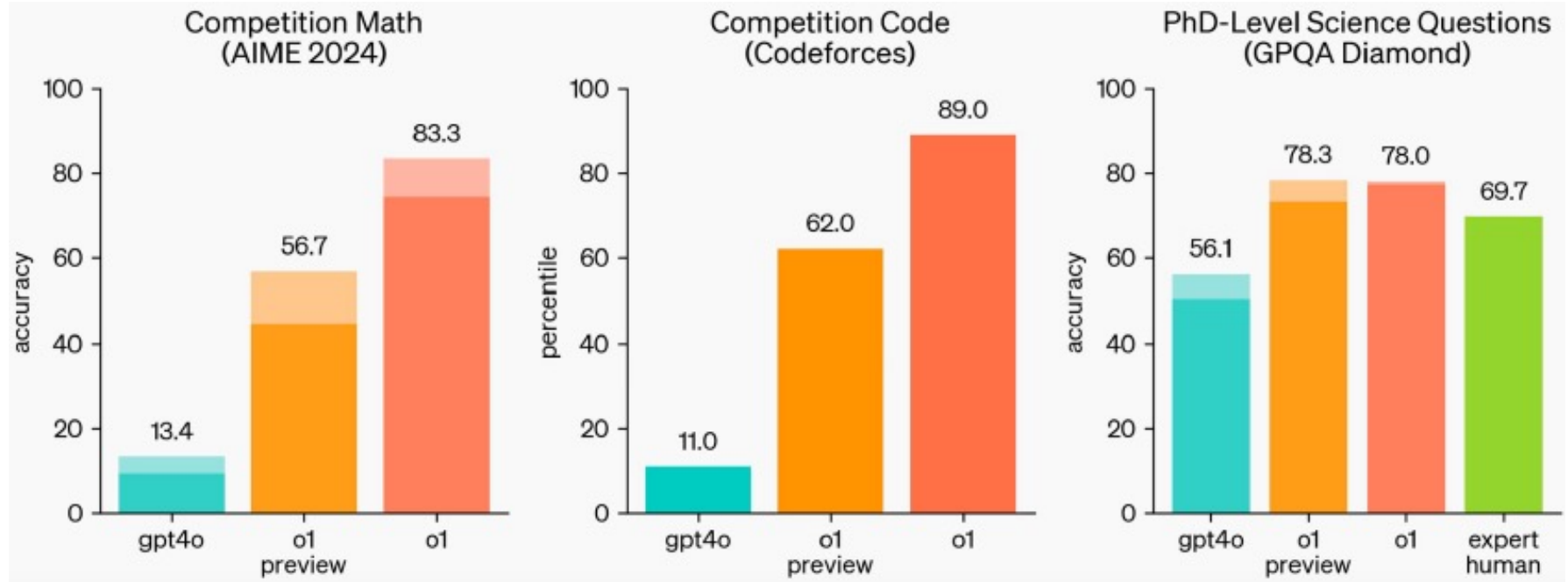
- + Deductive or inductive reasoning (e.g., riddles, math proofs)
- + Chain-of-thought reasoning (breaking down multi-step problems)
- + Complex decision-making tasks
- + Better generalization to novel problems

Bad at

- Fast and cheap responses (more inference time)
- Knowledge-based tasks (hallucination)
- Simple tasks (“overthinking”)

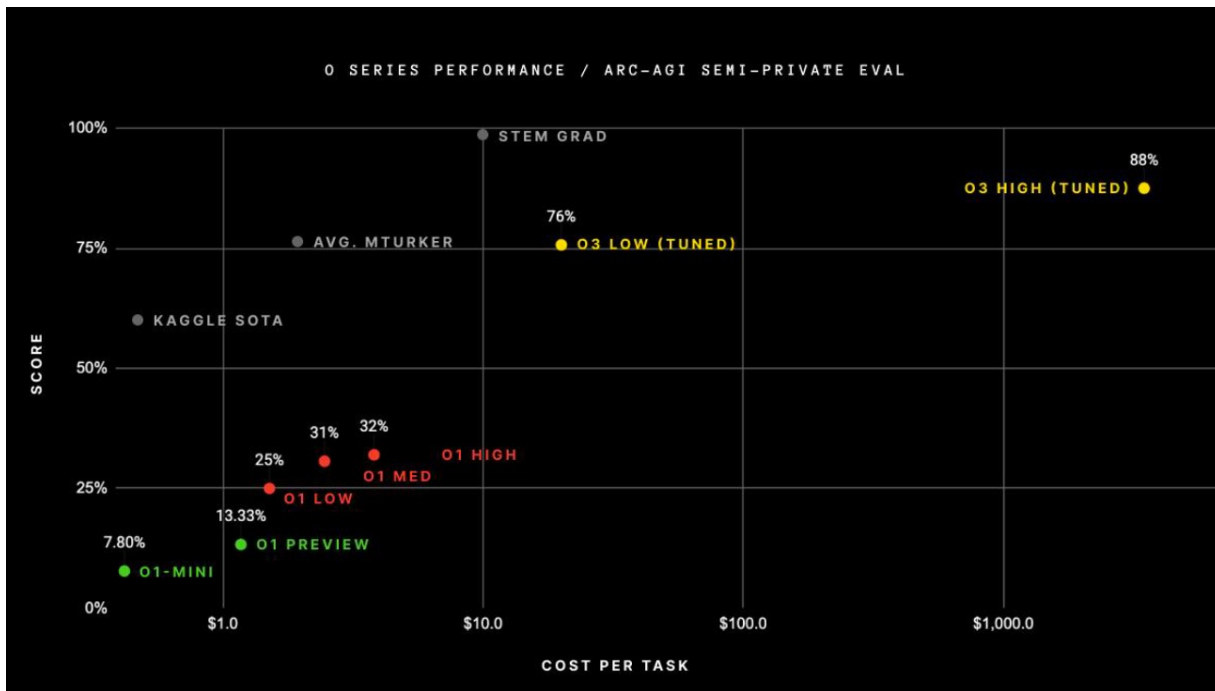


Highlight of LLMs in 2024: The Advancement of Reasoning Models



OpenAI o1 started to achieve impressive performance across various challenging reasoning tasks in math, coding, STEM, etc.

Effectiveness of Inference-Time Scaling for LLM Reasoning confirmed in 2024



- ❖ O3 achieved 87.5% accuracy on ARC-AGI, with >\$1k test-time cost to solve each task
- ❖ Other existing LLMs achieved <25% accuracy if no special inference-time techniques were used

Performance improves with more inference-time compute

Realizing LLM Reasoning via Inference-Time Scaling

- ❖ **Core Idea:** Trigger the LLM to generate one or more **LONG** Chain-of-Thought (CoT) Path of Exploration via one or more of the following:

- Few-Shot CoT Prompting
- Instruction Prompting
- Instruction Tuning
- Reinforcement Learning

Chain-of-Thought Prompting

Model Input

Q: 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?

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

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had $23 - 20 = 3$. They bought 6 more apples, so they have $3 + 6 = 9$. The answer is 9. ✓

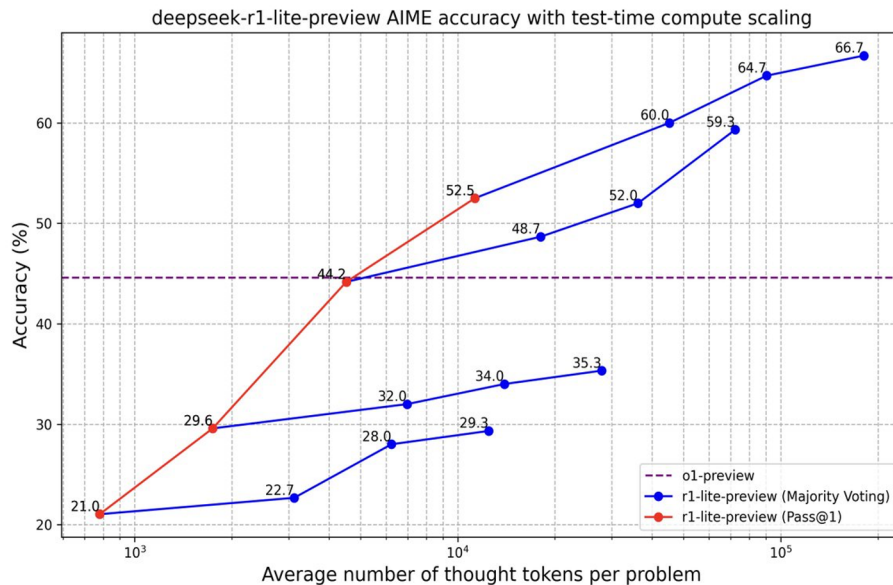
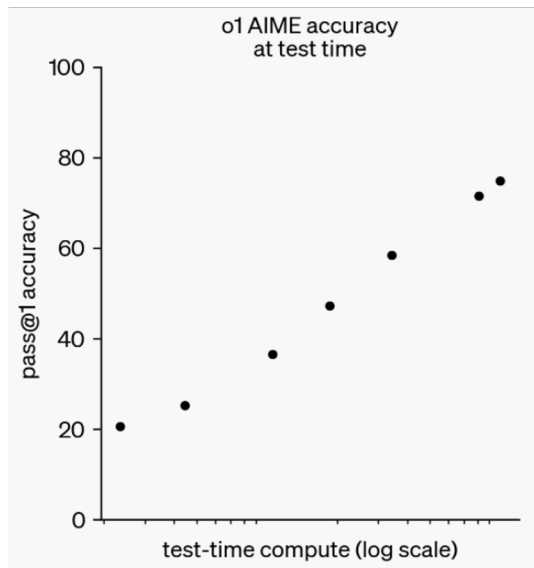
Sources:

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

Nye et al., Show Your Work: Scratchpads for Intermediate Computation with Language Models, 2021

Recent Work on Scaling Inference-time Compute

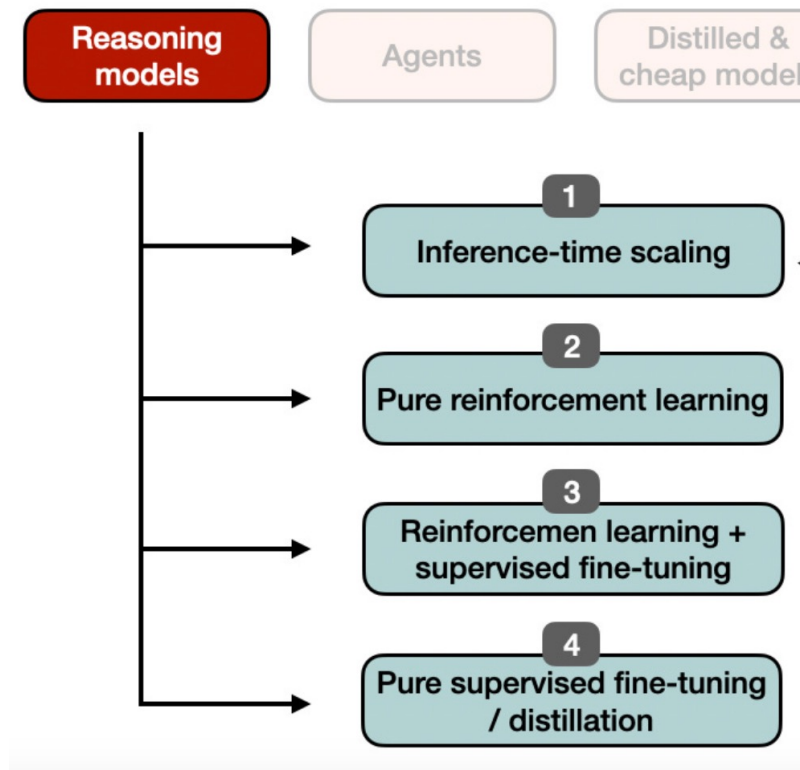
- GPTo1 / Deepseek R1 shows that:
 - **RL optimized Chain of Thought** can be a highly effective way to scaling test-time compute.



Realizing LLM Reasoning via Inference-Time Scaling

- ❖ **Core Idea:** Teach/ Force the LLM to generate one or more **LONG** Chain-of-Thought (CoT) Paths of Exploration using one or more of the following ways:

At least
Four different ways to
create a Reasoning
LLM



4 main ways to build or improve reasoning capabilities

1) Inference-time compute scaling

] **test-time compute**

2) Pure RL (e.g., DeepSeek R1 Zero)

3) RL + SFT (e.g., DeepSeek R1)

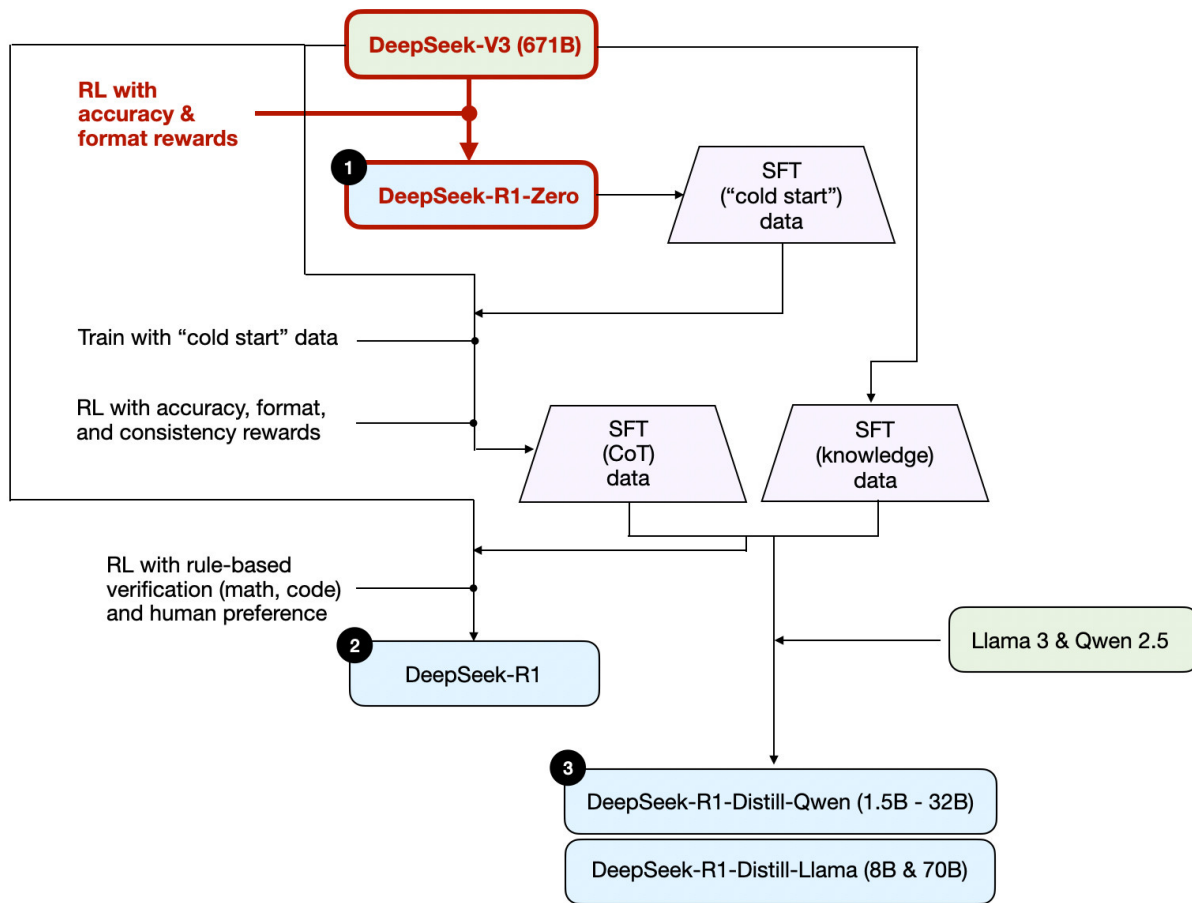
] **train-time compute**

4) pure SFT (e.g. s1, distilled DeepSeek R1)



Sebastian Raschka

Training DeepSeek-R1-Zero via “Pure” Reinforcement Learning



Training DeepSeek-R1-Zero via “Pure” RL: The “Aha” moment

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 $\dots$

$$\left(\sqrt{a - \sqrt{a + x}}\right)^2 = x^2 \implies a - \sqrt{a + x} = x^2.$$

Rearrange to isolate the inner square root term:

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

$\dots$

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 $\dots$

We started with the equation:

$$\sqrt{a - \sqrt{a + x}} = x$$

First, let's square both sides:

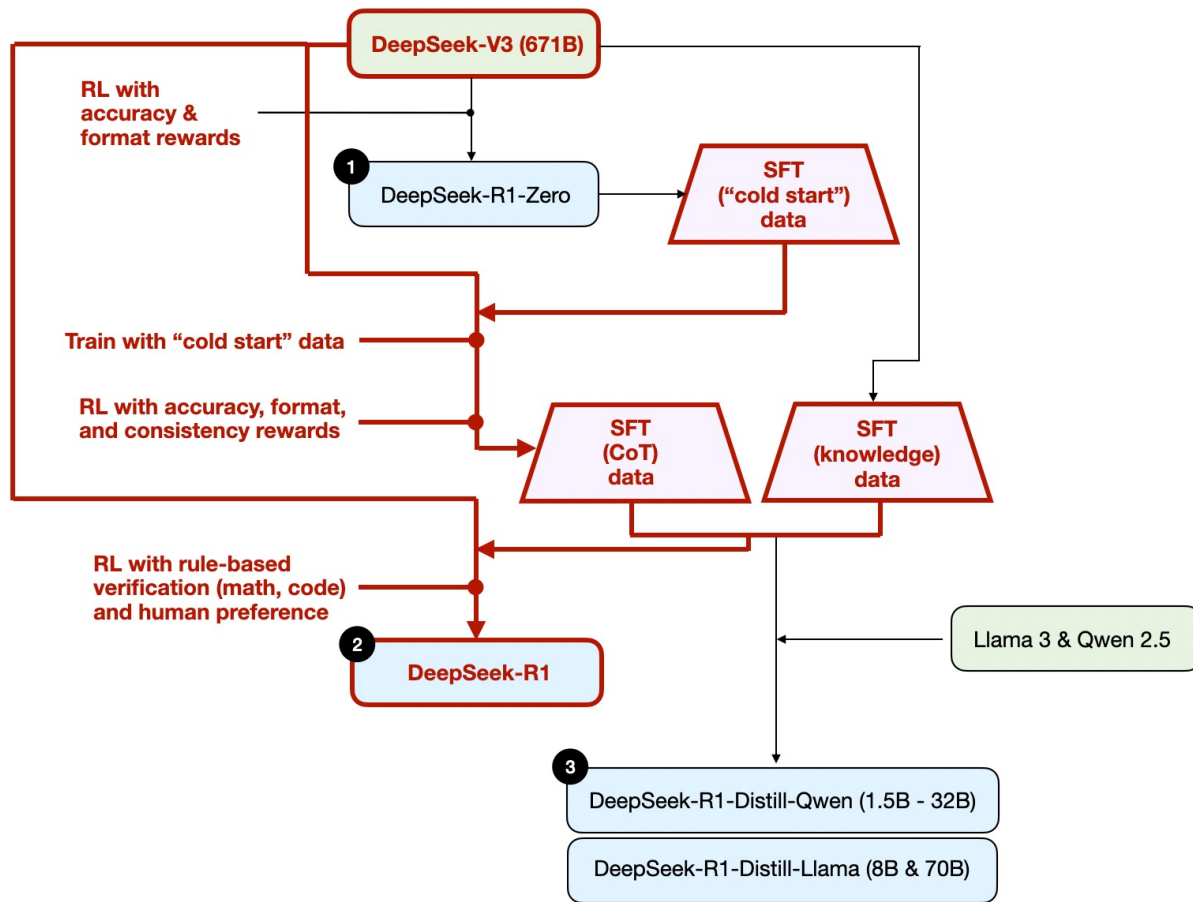
$$a - \sqrt{a + x} = x^2 \implies \sqrt{a + x} = a - x^2$$

Next, I could square both sides again, treating the equation: $\dots$

$\dots$

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.

Training DeepSeek-R1 via Supervised Fine-Tuning (SFT) + RL



Performance Comparison of DeepSeek-R1-Zero (Pure RL) via DeepSeek-R1 (SFT + RL)

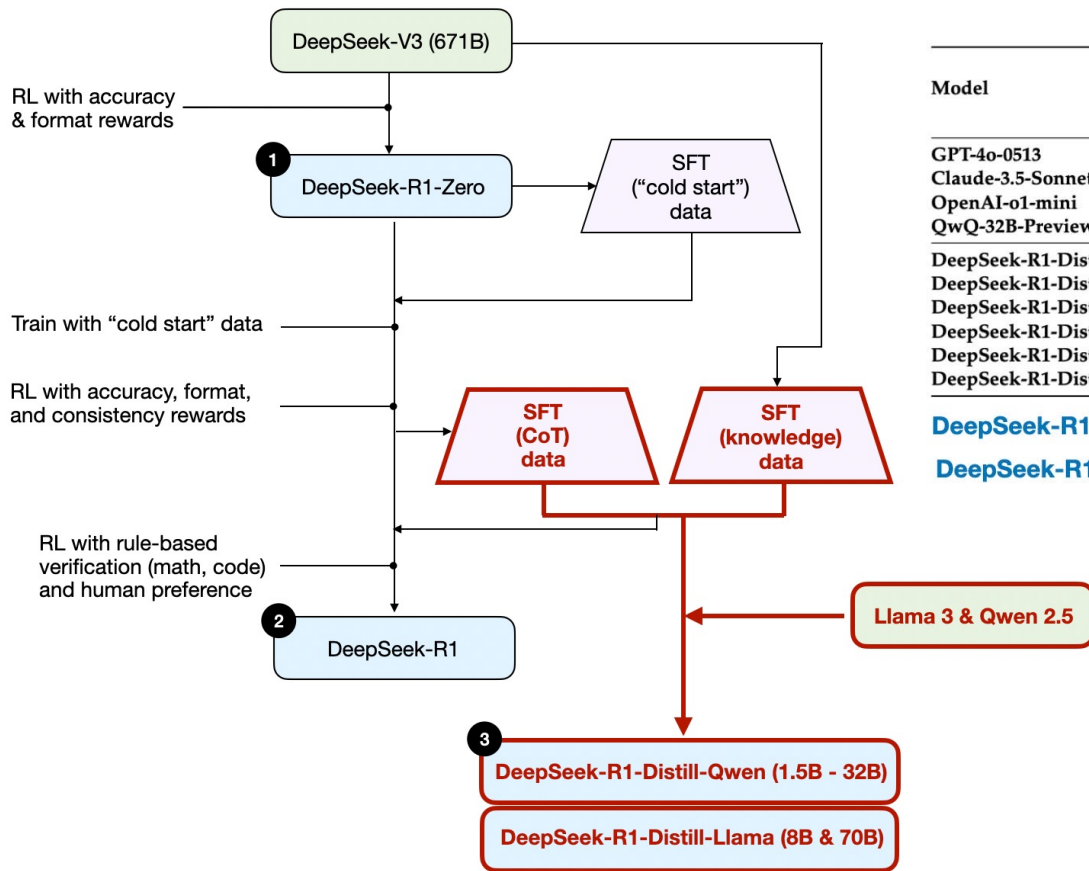
Model	Math benchmarks			Bio, physics & chemistry	Code benchmarks	
	AIME 2024		MATH-500	GPQA Diamond	LiveCode Bench	CodeForces
	pass@1	cons@64	pass@1	pass@1	pass@1	rating
OpenAI-o1-mini	63.6	80.0	90.0	60.0	53.8	1820
OpenAI-o1-0912	74.4	83.3	94.8	77.3	63.4	1843
DeepSeek-R1-Zero	71.0	86.7	95.9	73.3	50.0	1444
DeepSeek-R1	79.8		97.3	71.5	65.9	2029

Higher is better

RL only →

SFT + RL →

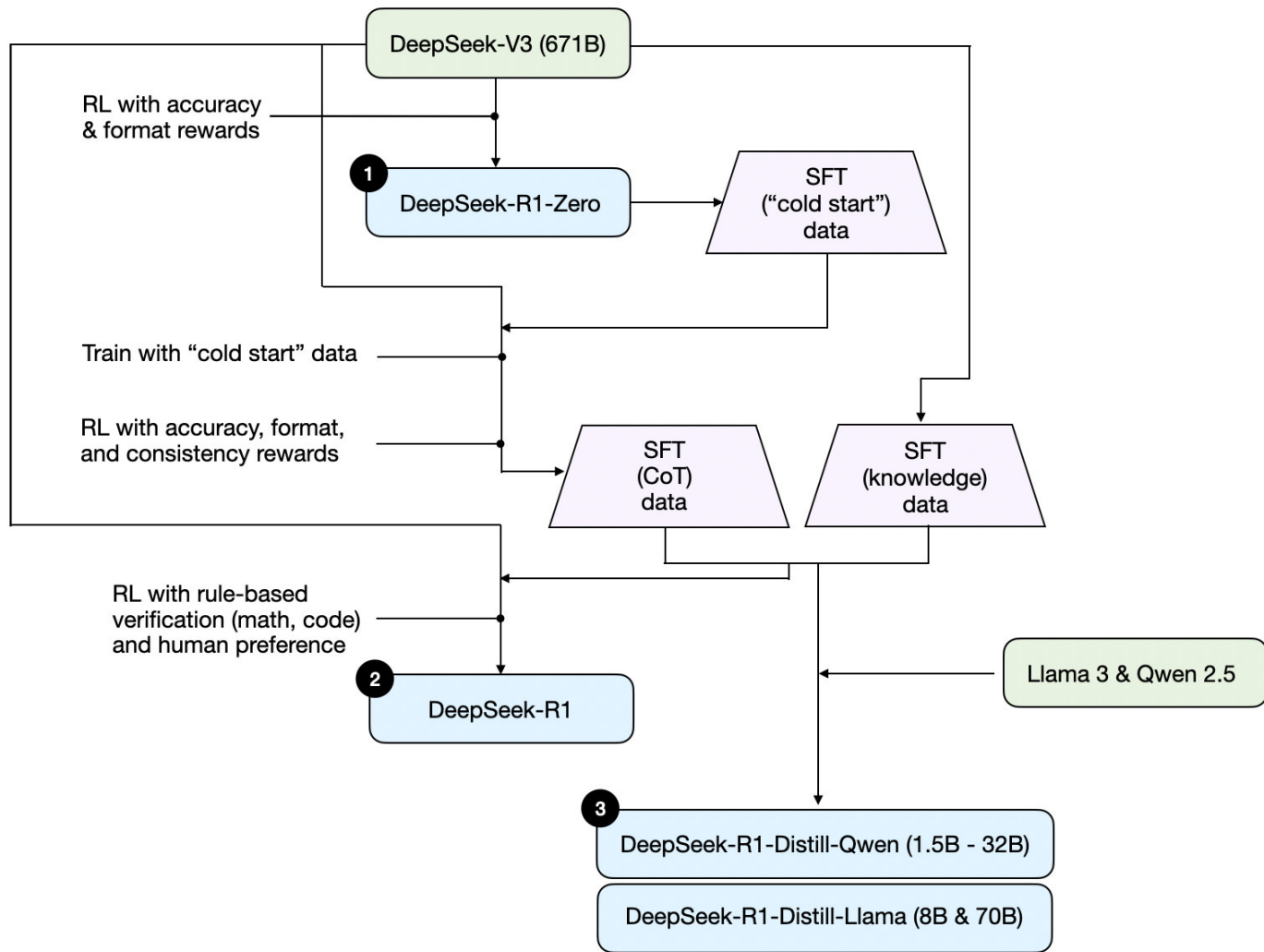
Training DeepSeek-R1-Distill* via SFT + Distillation



Model	AIME 2024		MATH-500	GPQA Diamond	LiveCode Bench	CodeForces
	pass@1	cons@64	pass@1	pass@1	pass@1	rating
GPT-4o-0513	9.3	13.4	74.6	49.9	32.9	759
Claude-3.5-Sonnet-1022	16.0	26.7	78.3	65.0	38.9	717
OpenAI-o1-mini	63.6	80.0	90.0	60.0	53.8	1820
QwQ-32B-Preview	50.0	60.0	90.6	54.5	41.9	1316
DeepSeek-R1-Distill-Qwen-1.5B	28.9	52.7	83.9	33.8	16.9	954
DeepSeek-R1-Distill-Qwen-7B	55.5	83.3	92.8	49.1	37.6	1189
DeepSeek-R1-Distill-Qwen-14B	69.7	80.0	93.9	59.1	53.1	1481
DeepSeek-R1-Distill-Qwen-32B	72.6	83.3	94.3	62.1	57.2	1691
DeepSeek-R1-Distill-Llama-8B	50.4	80.0	89.1	49.0	39.6	1205
DeepSeek-R1-Distill-Llama-70B	70.0	86.7	94.5	65.2	57.5	1633
DeepSeek-R1-Zero	71.0		95.9	73.3	50.0	1444
DeepSeek-R1	79.8		97.3	71.5	65.9	2029

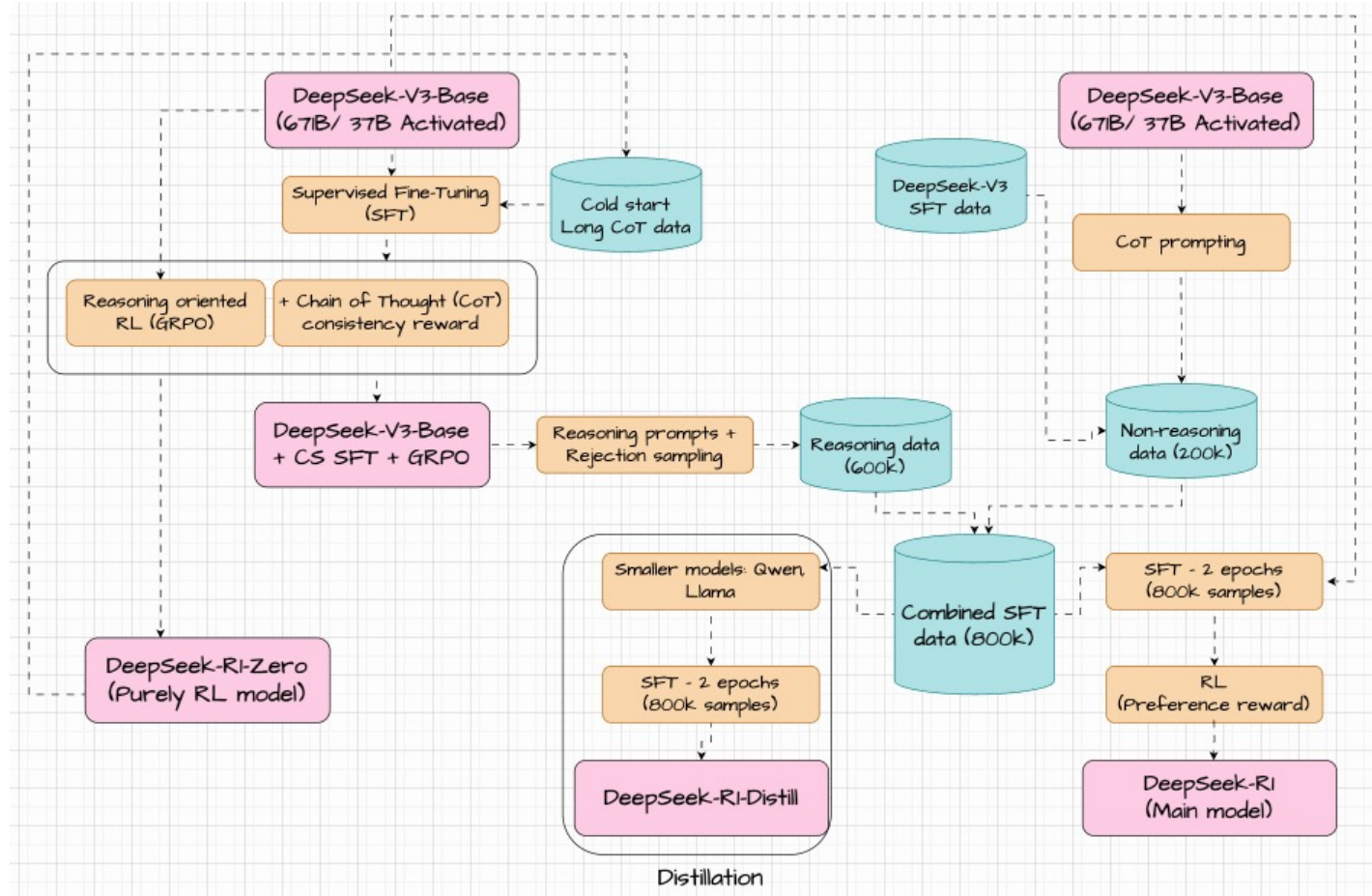
Summary

Training Process
of DeepSeek R1,
R1-Zero, and
R1-Distill* (all
are Reasoning
models)
from the
General Purpose
DeepSeekV3
model



More Details on the Training Process of

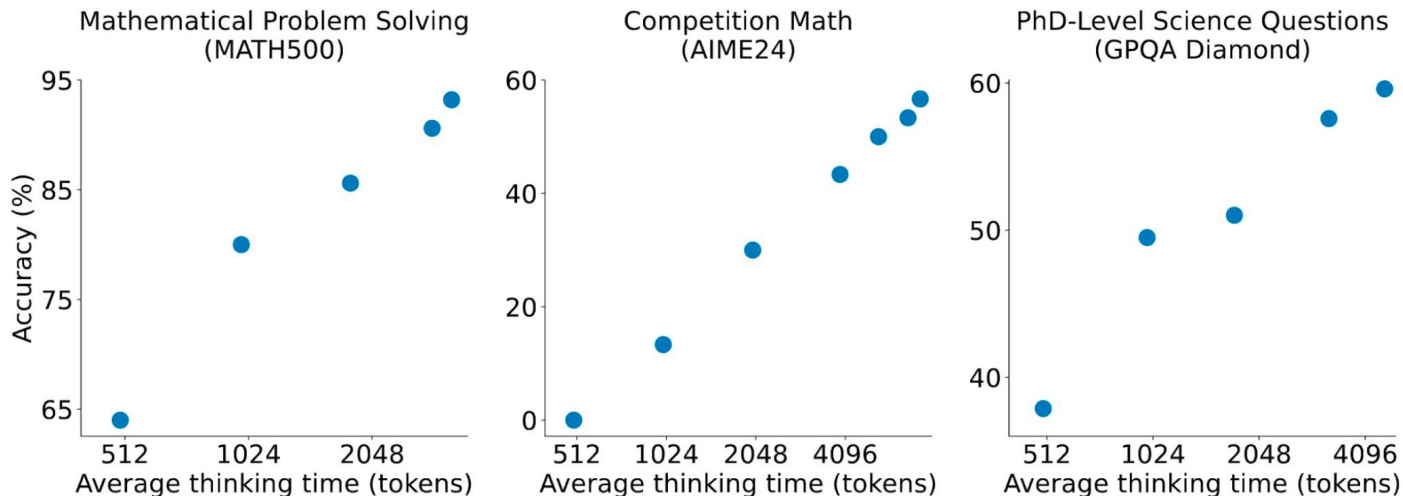
DeepSeek R1-zero, R1 (Reasoning models) from General Purpose DeepSeekV3 model



Minimal Recipe for Reasoning & Test-time Scaling

s1: Simple test-time scaling

Niklas Muennighoff* Zitong Yang* Weijia Shi* Xiang Lisa Li* Li Fei-Fei Hannaneh Hajishirzi
Luke Zettlemoyer Percy Liang Emmanuel Candès Tatsunori Hashimoto



Minimal Recipe for Reasoning & Test-time Scaling

Data

s1K

+

Test-time scaling

Budget forcing

=

s1

s1: Simple Test-Time Scaling [N. Muennihoff Jan/Mar 2025]

Procedure for creating the s1 model:

1. Create a curated SFT dataset, s1K, (a careful selection of rather difficult STEM questions, e.g. PhD Qualifying exam questions from top schools) containing 1000 training **Questions** paired with Answers + Reasoning Traces distilled from:
Gemini 2.0 Flash Thinking Experimental (for s1) or DeepSeek R1 (for s1.1)
2. Supervised Fine-tune the Qwen2.5-32B-Instruct LM on the s1K dataset (**No RL at all**)
3. Control the amount of Test-time compute consumed by the trained model via the following **Budget Forcing** mechanism:
 - if the model spends more thinking tokens than a preset limit, force it to end ;
 - if the model finishes too early (not thinking long enough), append **“Wait”** to the model’s current thinking trace and suppress the generation of the end-of-thinking token.

Creating the s1K dataset: Data Collection followed by Data filtering

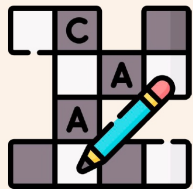
Data: Collect 59K questions

s1-prob

Show that the uniform distribution on $\llbracket[0,1]\rrbracket$ is not the convolution of two independent, identically distributed variables.

s1-teasers

Suppose you roll three fair 100-sided die. What is the expected value of the lowest roll?



Data Filtering

59K

Quality



52K

Difficulty



24K

Diversity



1K

Distilling Reasoning Traces & Answers

An often-repeated fun fact is that humans produce more power per unit volume than stars. If the sun were the same size, but it produced the same amount of power per unit volume as a human, what would its surface temperature be?...

s1: Google Gemini

The problem asks for the surface temperature of the sun if it had the same size but produced power per unit volume as a human.

First, let's find the power density of a human...

s1.1: DeepSeek r1

Okay, so there's this fun fact saying that humans produce more power per unit volume than stars. The question is asking if the Sun were the same size as a human (but with the same power per unit volume as a human), what would its surface temperature be? The answer choices are given, and I need to figure out which one is correct...

Composition of s1K dataset & Performance of the s1-32B model

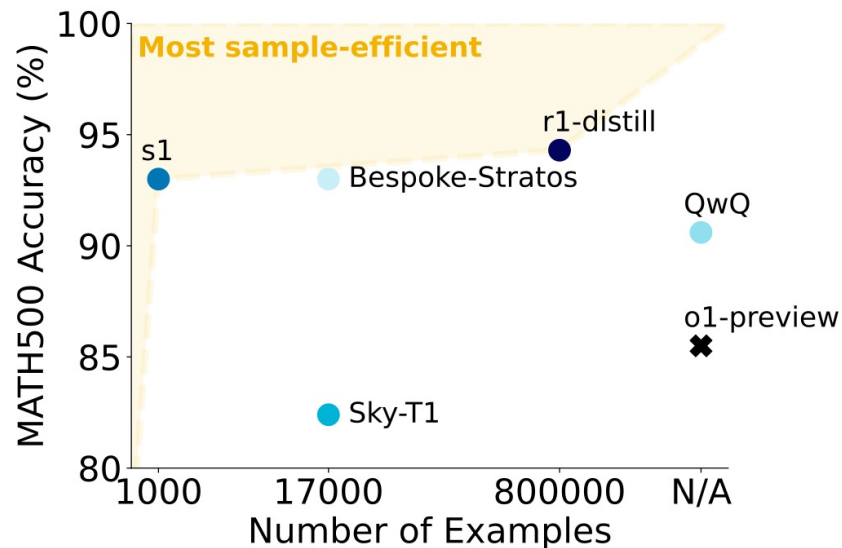
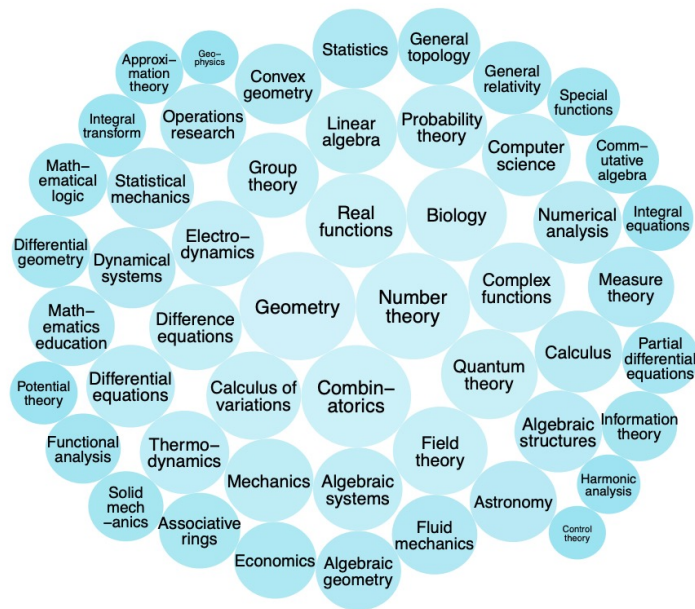


Figure 2. **s1K and s1-32B.** (left) **s1K** is a dataset of 1,000 high-quality, diverse, and difficult questions with reasoning traces. (right) **s1-32B**, a 32B parameter model finetuned on **s1K** is on the sample-efficiency frontier. See Table 1 for details on other models.

Test-time Scaling via Budget Forcing for the s1 model

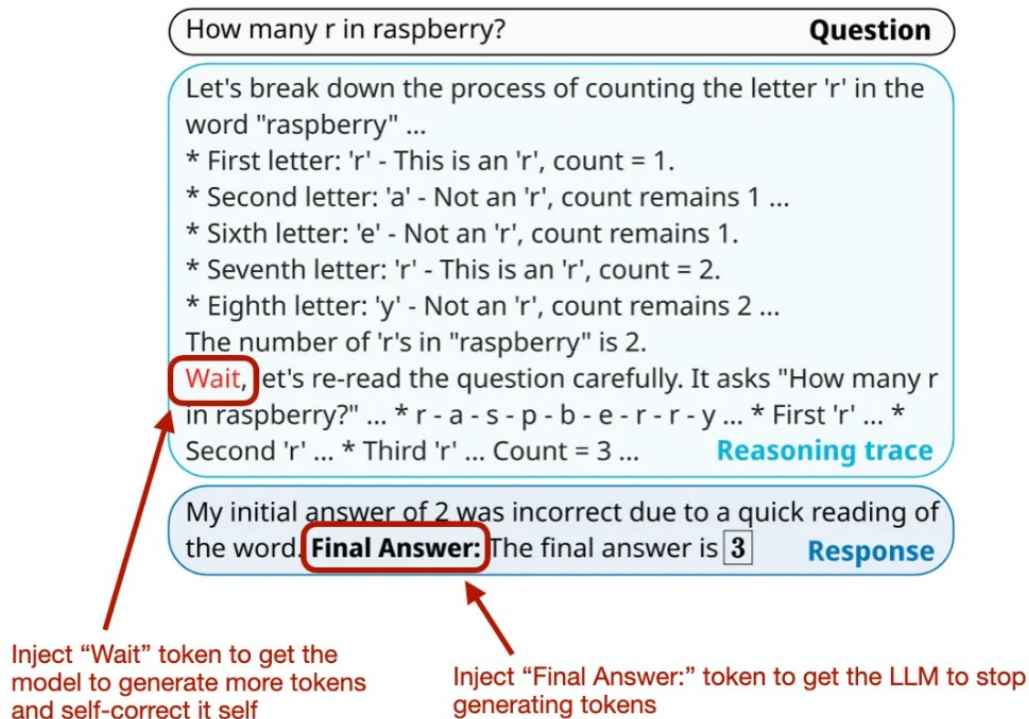
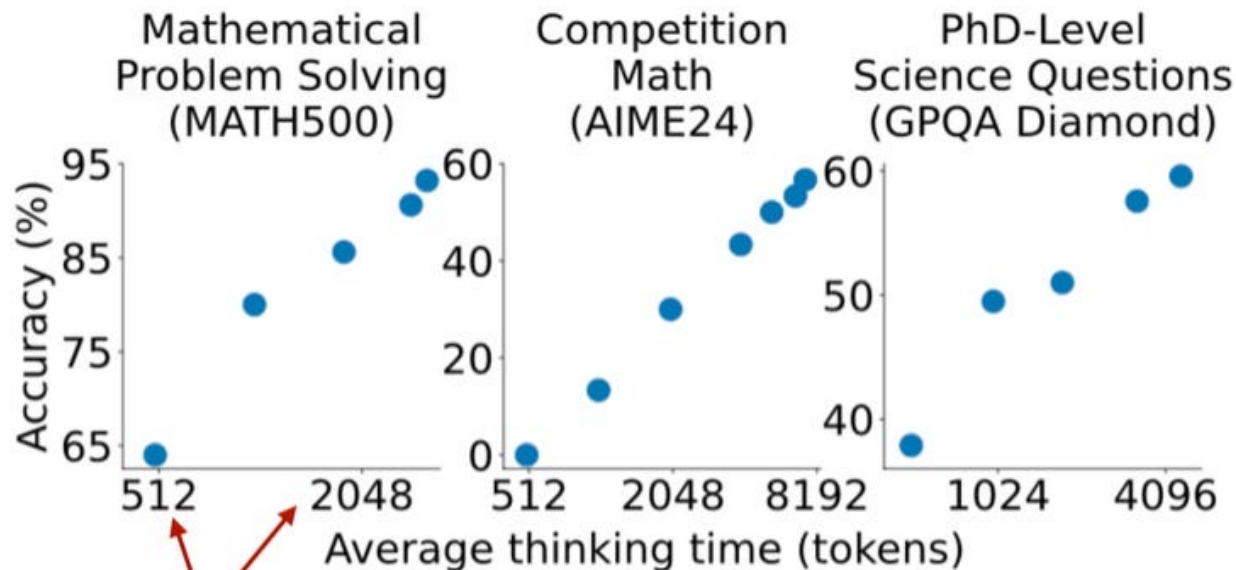


Illustration of "wait" token insertion to control the length of the output. Annotated figure from <https://arxiv.org/abs/2501.19393>.

Correlation b/w Tokens-generated and Response Accuracy by s1



There is a strong correlation between the number of generated tokens and the accuracy of the response

Ablation: Sequential and Parallel Test-time Scaling

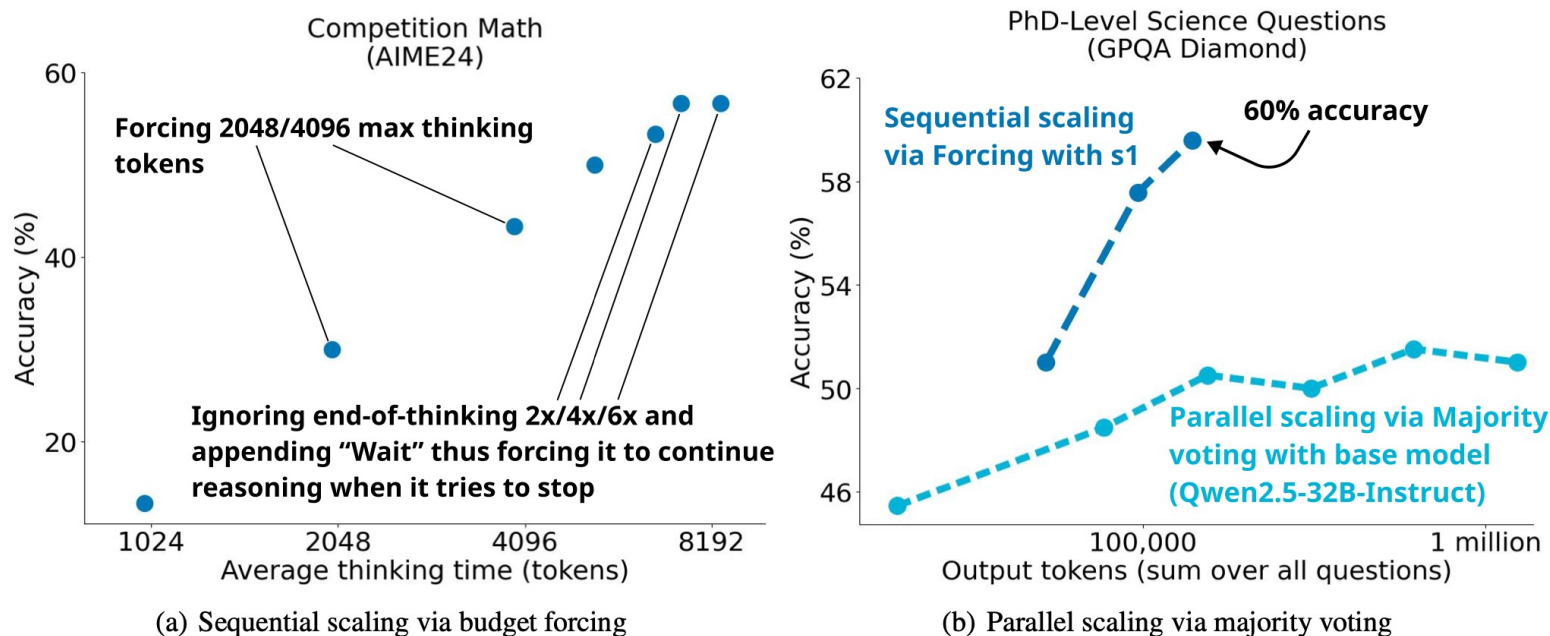


Figure 4. Sequential and parallel test-time scaling. (a): Budget forcing shows clear scaling trends and extrapolates to some extent. For the three rightmost dots, we prevent the model from stopping its thinking 2/4/6 times, each time appending “Wait” to its current reasoning trace. (b): For Qwen2.5-32B-Instruct we perform 64 evaluations for each sample with a temperature of 1 and visualize the performance when majority voting across 2, 4, 8, 16, 32, and 64 of these.

Ablation study on the choice of the Budget-Forcing string for s1

Table 4. Budget forcing extrapolation ablations. We compare ignoring the end-of-thinking delimiter twice and appending none or various strings.

Model	AIME 2024	MATH 500	GPQA Diamond
No extrapolation	50.0	93.0	57.6
2x without string	50.0	90.2	55.1
2x “Alternatively”	50.0	92.2	59.6
2x “Hmm”	50.0	93.0	59.6
2x “Wait”	53.3	93.0	59.6



“Wait” tokens perform better than “Hmm” tokens

Comparing s1-32B's Performance with other Reasoning models

Table 5. **s1-32B is an open and sample-efficient reasoning model.** We evaluate **s1-32B**, Qwen, and **Gemini** (some entries are unknown (N.A.), see §4). Other results are from the respective reports (Qwen et al., 2024; Team, 2024; OpenAI, 2024; 2025; DeepSeek-AI et al., 2025; Labs, 2025; Team, 2025) except for AIME 2025 (Ye et al., 2025b). # ex. = number examples used for reasoning finetuning; BF = budget forcing.

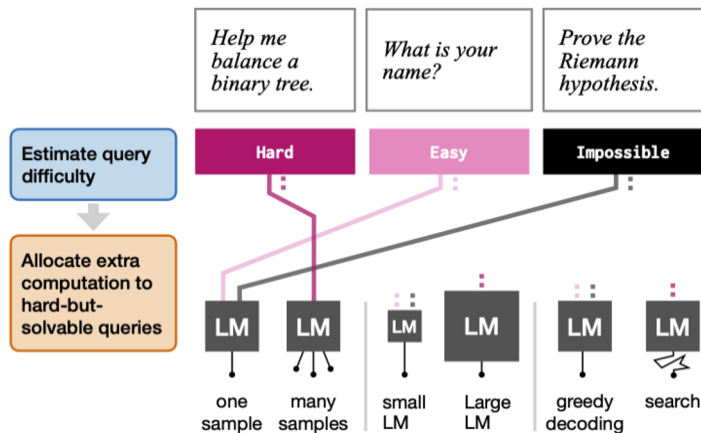
Model	# Examples	MATH500	GPQA	AIME 2024	AIME 2025
API only					
o3-mini-low	N/A	95.8	70.6	56.3	42.1
o3-mini-medium	N/A	97.3	76.8	75.8	70.4
o3-mini-high	N/A	97.9	79.7	83.8	80.9
Open Weights					
QwQ-32B	N.A.	90.6	54.5	46.7	32.7
r1	≫800K	07.3	71.5	79.8	70.0
r1-distill-Llama-70B	800K	94.5	65.2	57.1	56.3
r1-distill-Qwen-14B	800K	93.9	59.1	61.7	48.0
r1-distill-Qwen-32B	800K	94.3	62.1	58.3	49.6
Open Weights and Open Data					
LIMO	817	94.8	66.7	56.3	44.6
s1 w/o BF	1K	92.6	56.6	50.0	26.7
s1 with Budget Forcing “Wait” 1x	1K	92.8	59.6	53.3	30.0
s1 with Budget Forcing “Wait” 2x	1K	93.0	59.6	53.3	33.3
s1 with Budget Forcing “Wait” 4x	1K	92.2	58.6	56.7	36.7
s1.1 w/o BF	1K	94.4	60.6	56.7	50.0
s1.1 with Budget Forcing “Wait” 1x	1K	95.4	62.6	56.7	50.0
s1.1 with Budget Forcing “Wait” 2x	1K	95.4	63.6	56.7	50.0

Other Recent Research on Scaling Inference-time Scaling

Recent Work on how to determine Inference-time Compute

LEARNING HOW HARD TO THINK: INPUT-ADAPTIVE ALLOCATION OF LM COMPUTATION

Mehul Damani* Idan Shenfeld Andi Peng Andreea Bobu Jacob Andreas
Massachusetts Institute of Technology



❖ Recent work shows that models can be [fine-tuned](#) to [cheaply assess difficulty](#).

Less (but Good Training Examples) is More for Reasoning

LIMO: Less is More for Reasoning

Yixin Ye* Zhen Huang* Yang Xiao Ethan Chern Shijie Xia Pengfei Liu†

SJTU, SII, GAIR

Abstract

We present a fundamental *discovery* that challenges our understanding of how complex reasoning emerges in large language models. While conventional wisdom suggests that sophisticated reasoning tasks demand extensive training data (often $> 100,000$ examples), we demonstrate a striking phenomenon: complex mathematical reasoning abilities can be effectively elicited with *surprisingly* few examples. This finding challenges not only the assumption of massive data requirements but also the common belief that supervised fine-tuning primarily leads to memorization rather than generalization. Through comprehensive experiments, our proposed model LIMO demonstrates unprecedented performance and efficiency in mathematical reasoning. With merely **817** curated training samples, LIMO achieves **57.1%** accuracy on the highly challenging AIME benchmark and **94.8%** on MATH, improving the performance of previous strong SFT-based models from 6.5% to 57.1% on AIME and from 59.2% to 94.8% on MATH, while only using 1% of the training data required by previous approaches. Most remarkably, LIMO demonstrates exceptional out-of-distribution generalization, achieving **40.5%** absolute improvement across **10** diverse benchmarks, outperforming models trained on 100x more data, directly challenging the prevailing notion that SFT inherently leads to memorization rather than generalization. Synthesizing these pioneering results, we propose the **Less-Is-More Reasoning Hypothesis (LIMO Hypothesis)**: *In foundation models where domain knowledge has been comprehensively encoded during pre-training, sophisticated reasoning capabilities can emerge through minimal but precisely orchestrated demonstrations of cognitive processes.* This hypothesis posits that the *elicitation threshold* for complex reasoning is not inherently bounded by the complexity of the target reasoning task, but fundamentally determined by two key factors: (1) the completeness of the model’s encoded knowledge foundation during pre-training, and (2) the effectiveness of post-training examples, which serve as “cognitive templates” that show the model how to effectively utilize its existing knowledge base to solve complex reasoning tasks. To facilitate reproducibility and future research in data-efficient reasoning, we release LIMO as a comprehensive open-source suite at <https://github.com/GAIR-NLP/LIMO>.

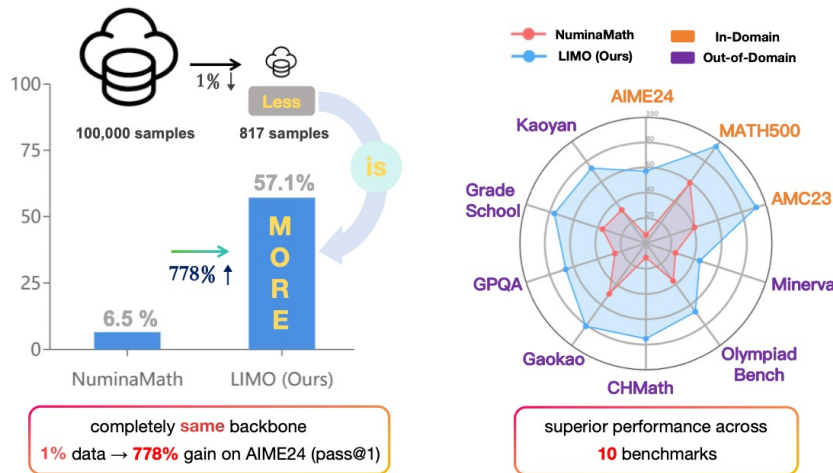


Figure 1: LIMO achieves substantial improvement over NuminaMath with fewer samples while excelling on diverse mathematical and multi-discipline benchmarks.

LIMO vs. Reinforcement Learning Scaling

Table 2: Comparative Analysis of LIMO and RL Scaling Approaches

Aspect	RL Scaling (e.g., o1, R1)	LIMO
First Principle	An implementation of the general principle: searching for optimal reasoning trajectories through RL	The fundamental principle: reasoning capabilities exist and need to be activated by high-quality reasoning trajectories
Solution Nature	Discovers reasoning trajectories through extensive RL-based exploration	Directly constructs high-quality reasoning trajectories based on cognitive understanding
Core Challenge	How to efficiently search for effective reasoning trajectories in a large solution space	How to identify and construct optimal reasoning trajectories that activate existing capabilities
Methodology	Implicit trajectory discovery through large-scale RL optimization	Explicit trajectory design through cognitive templates
Search Strategy	Broad exploration of solution space using computational resources	Targeted exploration guided by cognitive principles
Resource Efficiency	Resource-intensive search process	Resource-efficient direct construction
Generalization	Through extensive sampling of trajectory space	Through understanding of fundamental reasoning patterns