Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters [arXiv 24.08]

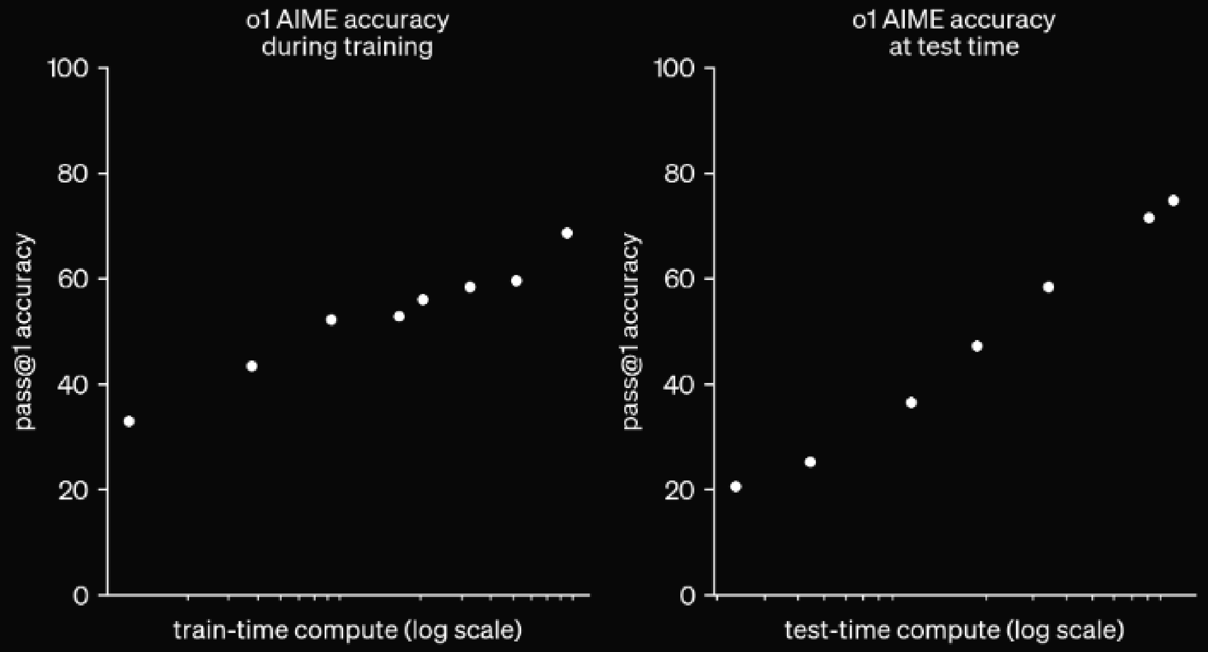
TL; DR.
Explores two main strategies (PRM & Refining the Proposal Distribution) for scaling LLM reasoning at test-time.

Presented by:
Jiaxi Li

What is Scaling Law

- For training OpenAI o1
  - Scaling Law for both train-time and **test-time**.

Our large-scale reinforcement learning algorithm teaches the model how to think productively using its chain of thought in a highly data-efficient training process. We have found that the performance of o1 consistently improves with more reinforcement learning (train-time compute) and with more time spent thinking (test-time compute). The constraints on scaling this approach differ substantially from those of LLM pretraining, and we are continuing to investigate them.



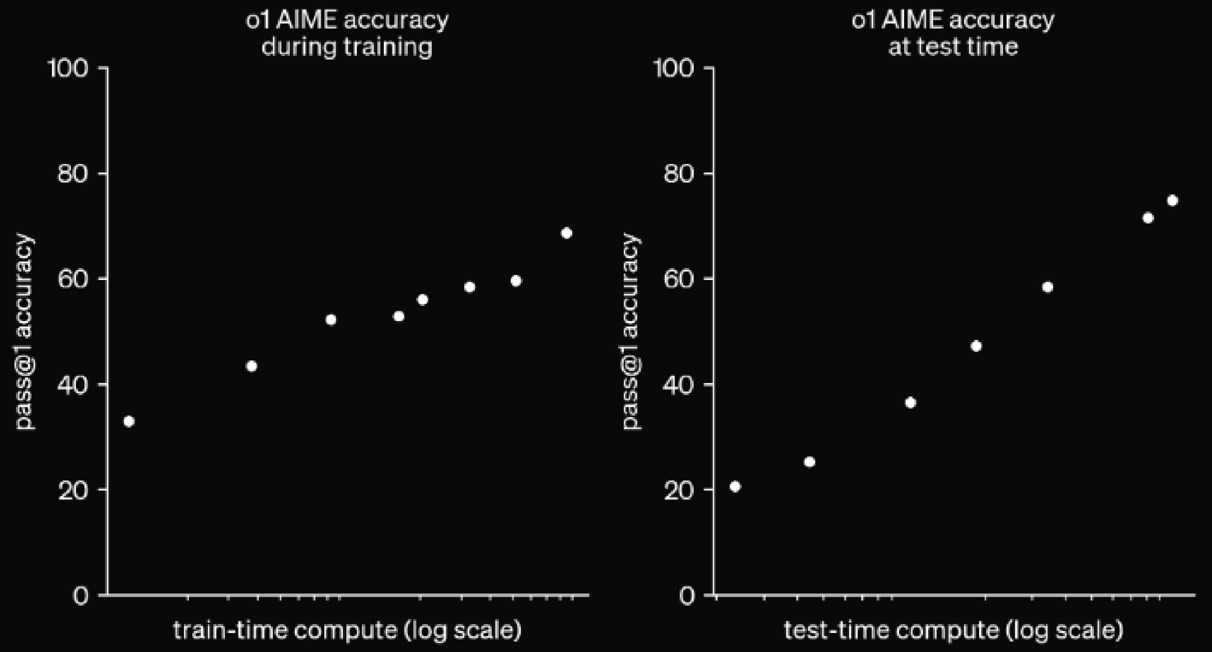
o1 performance smoothly improves with both train-time and test-time compute

[1] OpenAI "Learning to Reason with LLMs". 2024.09

What is Scaling Law

- For training OpenAI o1
  - Scaling Law for both train-time and **test-time**.
- Question

Our large-scale reinforcement learning algorithm teaches the model how to think productively using its chain of thought in a highly data-efficient training process. We have found that the performance of o1 consistently improves with more reinforcement learning (train-time compute) and with more time spent thinking (test-time compute). The constraints on scaling this approach differ substantially from those of LLM pretraining, and we are continuing to investigate them.



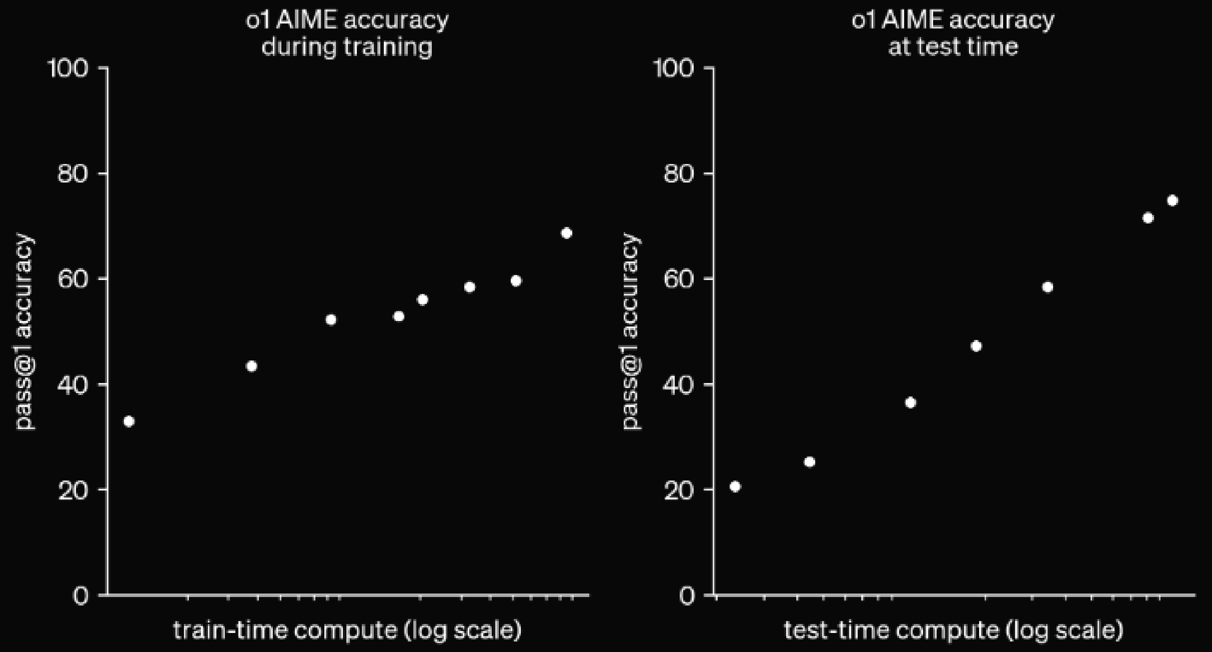
o1 performance smoothly improves with both train-time and test-time compute

[1] OpenAI "Learning to Reason with LLMs". 2024.09

What is Scaling Law

- For training OpenAI o1
  - Scaling Law for both train-time and **test-time**.
- Question
  - What do they mean by "test-time compute"?
    And how to scale up "test-time compute"?

Our large-scale reinforcement learning algorithm teaches the model how to think productively using its chain of thought in a highly data-efficient training process. We have found that the performance of o1 consistently improves with more reinforcement learning (train-time compute) and with more time spent thinking (test-time compute). The constraints on scaling this approach differ substantially from those of LLM pretraining, and we are continuing to investigate them.

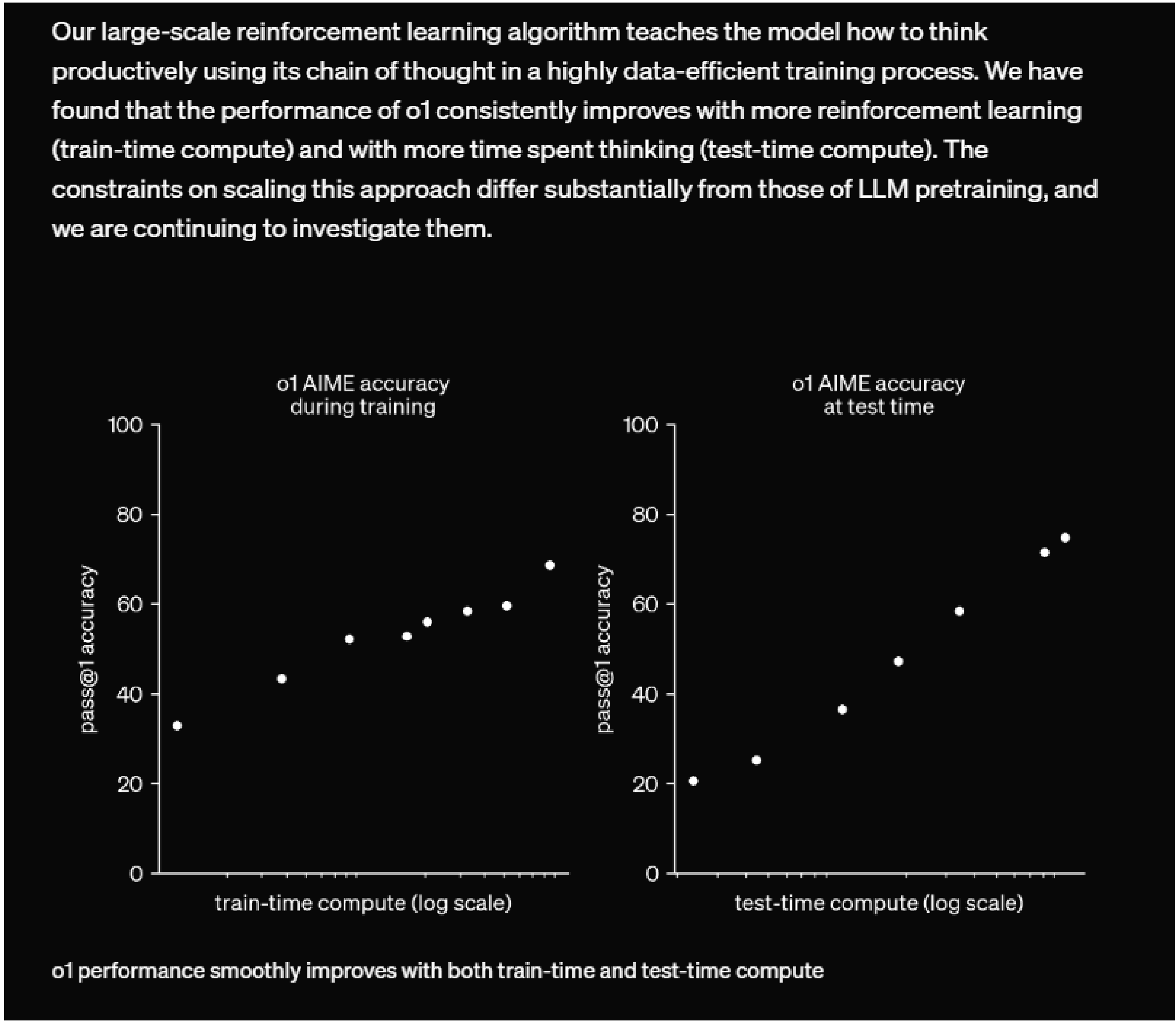


o1 performance smoothly improves with both train-time and test-time compute

[1] OpenAI "Learning to Reason with LLMs". 2024.09

What is Scaling Law

- For training OpenAI o1
  - Scaling Law for both train-time and **test-time**.
- Question
  - What do they mean by "test-time compute"?
    And how to scale up "test-time compute"?
- A shift from "system-1" to "system-2" reasoning.



[1] OpenAI "Learning to Reason with LLMs". 2024.09

How to scale up test-time compute?

- For optimizing input (prompting)

How to scale up test-time compute?

- For optimizing input (prompting)
    - Basic prompting techniques

How to scale up test-time compute?

- For optimizing input (prompting)
  - Basic prompting techniques
    - Few-shot prompting

How to scale up test-time compute?

- For optimizing input (prompting)
  - Basic prompting techniques
    - Few-shot prompting
    - CoT prompting

How to scale up test-time compute?

- For optimizing input (prompting)
  - Basic prompting techniques
    - Few-shot prompting
    - CoT prompting
  - Learning to prompt (using neural networks)

How to scale up test-time compute?

- For optimizing input (prompting)
  - Basic prompting techniques
    - Few-shot prompting
    - CoT prompting
  - Learning to prompt (using neural networks)
    - RLPrompt[2]

How to scale up test-time compute?

- For optimizing input (prompting)
  - Basic prompting techniques
    - Few-shot prompting
    - CoT prompting
  - Learning to prompt (using neural networks)
    - RLPrompt[2]
    - DSPy[3]

How to scale up test-time compute?

- For optimizing input (prompting)
  - Basic prompting techniques
    - Few-shot prompting
    - CoT prompting
  - Learning to prompt (using neural networks)
    - RLPrompt[2]
    - DSPy[3]
      - Already built into python packages and widely used

How to scale up test-time compute?

- For optimizing input (prompting)
  - Basic prompting techniques
    - Few-shot prompting
    - CoT prompting
  - Learning to prompt (using neural networks)
    - RLPrompt[2]
    - DSPy[3]
      - Already built into python packages and widely used

[2] Deng et al., "RLPrompt: Optimizing Discrete Text Prompts with Reinforcement Learning" EMNLP 2022

How to scale up test-time compute?

- For optimizing input (prompting)
  - Basic prompting techniques
    - Few-shot prompting
    - CoT prompting
  - Learning to prompt (using neural networks)
    - RLPrompt[2]
    - DSPy[3]
      - Already built into python packages and widely used

[2] Deng et al., "RLPrompt: Optimizing Discrete Text Prompts with Reinforcement Learning" EMNLP 2022

[3] Khattab et al., "DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines" R0-FoMo@NeurIPS 2023

How to scale up test-time compute?

- For optimizing input (prompting)
  - Basic prompting techniques
    - Few-shot prompting
    - CoT prompting
  - Learning to prompt (using neural networks)
    - RLPrompt[2]
    - DSPy[3]
      - Already built into python packages and widely used
  - And many other techniques for optimizing prompts…

[2] Deng et al., "RLPrompt: Optimizing Discrete Text Prompts with Reinforcement Learning" EMNLP 2022

[3] Khattab et al., "DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines" R0-FoMo@NeurIPS 2023

How to scale up test-time compute?

- For refining output distribution
  - How to let LLMs generate better CoT rationales?
    - SFT works.
    - SFT with collected CoT rationales can let LLM generate better reasoning traces.

How to scale up test-time compute?

- For refining output distribution
  - How to let LLMs generate better CoT rationales?
    - SFT works.
    - SFT with collected CoT rationales can let LLM generate better reasoning traces.
  - Take a step further, how to let LLM keep revising its CoT rationales and gradually approach a more reasonable answer?

How to scale up test-time compute?

- For refining output distribution
  - How to let LLMs generate better CoT rationales?
    - SFT works.
    - SFT with collected CoT rationales can let LLM generate better reasoning traces.
  - Take a step further, how to let LLM keep revising its CoT rationales and gradually approach a more reasonable answer?
    - Tree-of-Thought

How to scale up test-time compute?

- For refining output distribution
  - How to let LLMs generate better CoT rationales?
    - SFT works.
    - SFT with collected CoT rationales can let LLM generate better reasoning traces.
  - Take a step further, how to let LLM keep revising its CoT rationales and gradually approach a more reasonable answer?
    - Tree-of-Thought
    - Monte-Carlo Tree Search

How to scale up test-time compute?

- For refining output distribution
  - How to let LLMs generate better CoT rationales?
    - SFT works.
    - SFT with collected CoT rationales can let LLM generate better reasoning traces.
  - Take a step further, how to let LLM keep revising its CoT rationales and gradually approach a more reasonable answer?
    - Tree-of-Thought
    - Monte-Carlo Tree Search
    - …

How to scale up test-time compute?

- For refining output distribution
  - How to let LLMs generate better CoT rationales?
    - SFT works.
    - SFT with collected CoT rationales can let LLM generate better reasoning traces.
  - Take a step further, how to let LLM keep revising its CoT rationales and gradually approach a more reasonable answer?
    - Tree-of-Thought
    - Monte-Carlo Tree Search
    - …
  - Both of them contribute to training a **verifier** to help refine the output distribution at test-time.

The scaling-up strategies for test-time

- Scaling Test-Time Compute via Verifiers
  - Training verifiers to search
  - Search Methods Against a verifier
- Refining the Proposal Distribution
  - Parallel Sampling v.s. Sequential Revisions
  - Trading off between them

The scaling-up strategies for test-time

- Scaling Test-Time Compute via Verifiers
  - Training verifiers to search
  - Search Methods Against a verifier
- Refining the Proposal Distribution
  - Parallel Sampling v.s. Sequential Revisions
  - Trading off between them

- [Q] Aren't they talking about test-time? Why are they still training?

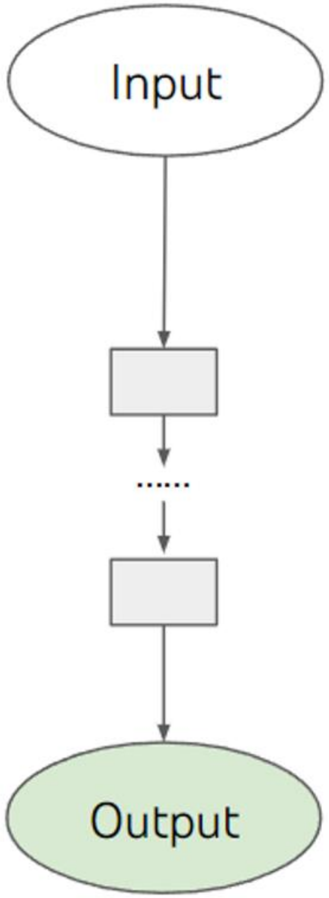
The scaling-up strategies for test-time

- Scaling Test-Time Compute via Verifiers
  - Training verifiers to search
  - Search Methods Against a verifier
- Refining the Proposal Distribution
  - Parallel Sampling v.s. Sequential Revisions
  - Trading off between them

- [Q] Aren't they talking about test-time? Why are they still training?
  - To scale up compute at test-time, we cannot do it without **post-training**.

Scaling Test-Time Compute via Verifiers

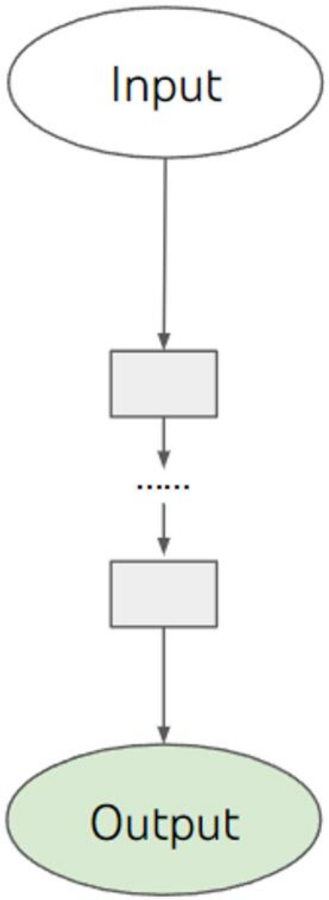
- So what are verifiers?



A CoT rationale

Scaling Test-Time Compute via Verifiers

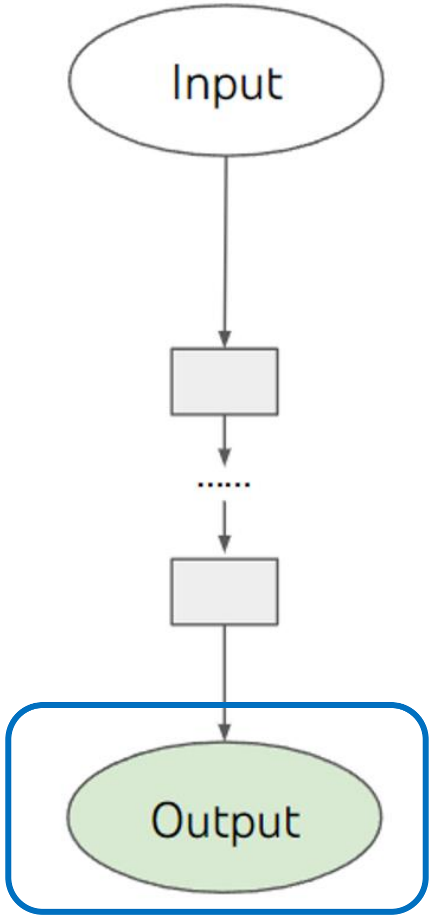
- So what are verifiers?
  - ORM: Outcome-supervised Reward Model



A CoT rationale

Scaling Test-Time Compute via Verifiers

- So what are verifiers?
  - ORM: Outcome-supervised Reward Model

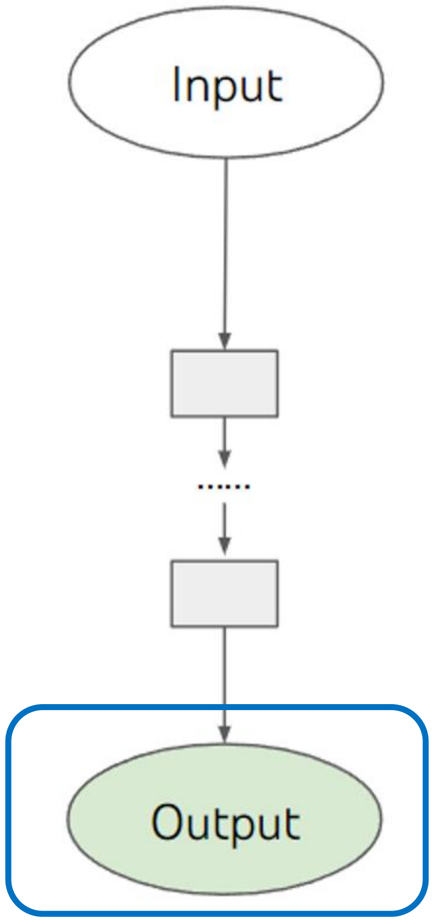


A CoT rationale

Scaling Test-Time Compute via Verifiers

- So what are verifiers?
  - ORM: Outcome-supervised Reward Model

ORM <= Output + Label

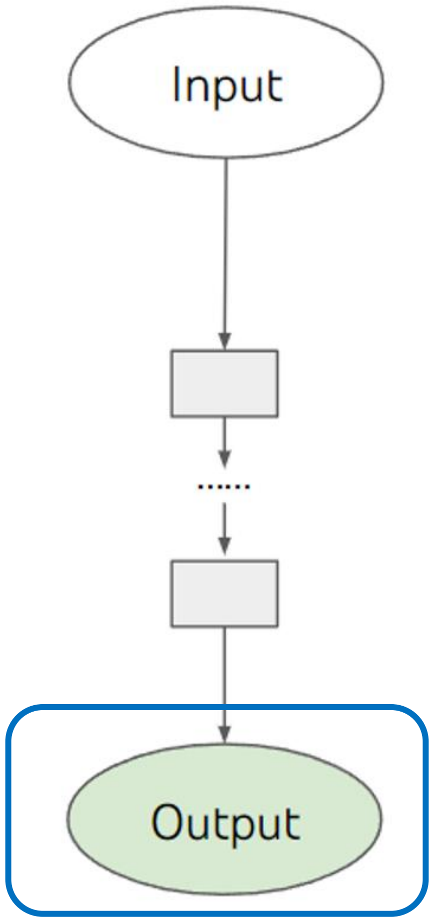


A CoT rationale

Scaling Test-Time Compute via Verifiers

- So what are verifiers?
  - ORM: Outcome-supervised Reward Model
  - PRM: Process-supervised Reward Model

ORM <= Output + Label



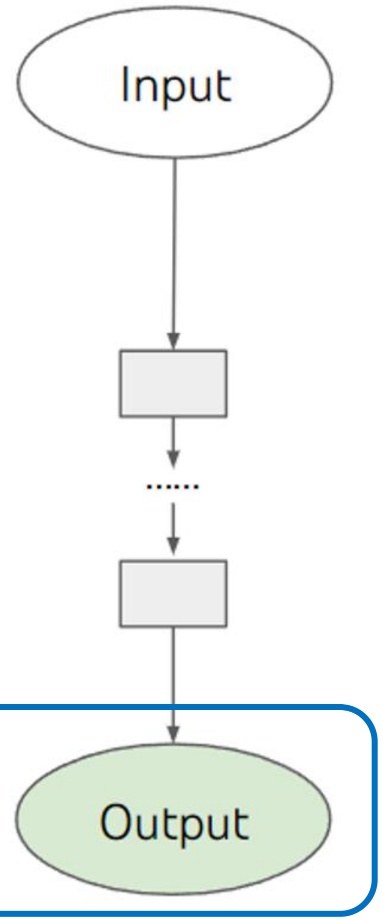
A CoT rationale

Scaling Test-Time Compute via Verifiers

- So what are verifiers?
  - ORM: Outcome-supervised Reward Model
  - PRM: Process-supervised Reward Model

PRM <= Output + Label + supervision of rationales

ORM <= Output + Label



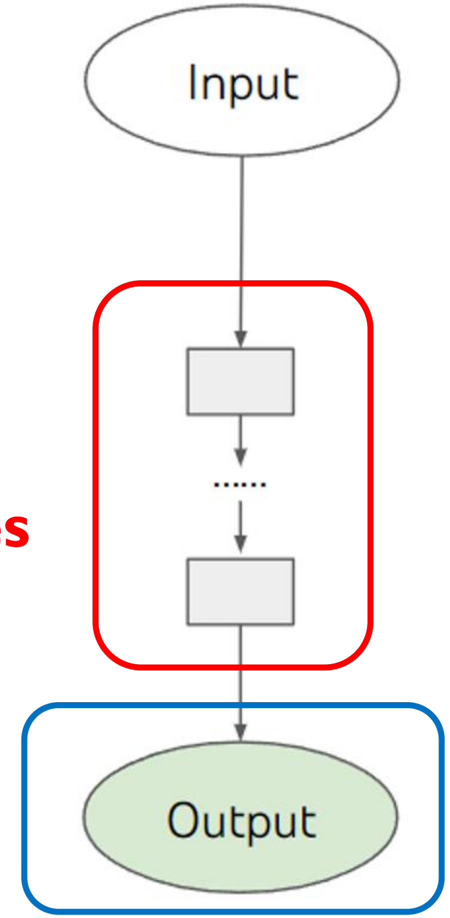
A CoT rationale

Scaling Test-Time Compute via Verifiers

- So what are verifiers?
  - ORM: Outcome-supervised Reward Model
  - PRM: Process-supervised Reward Model

PRM <= Output + Label + supervision of rationales

ORM <= Output + Label



A CoT rationale

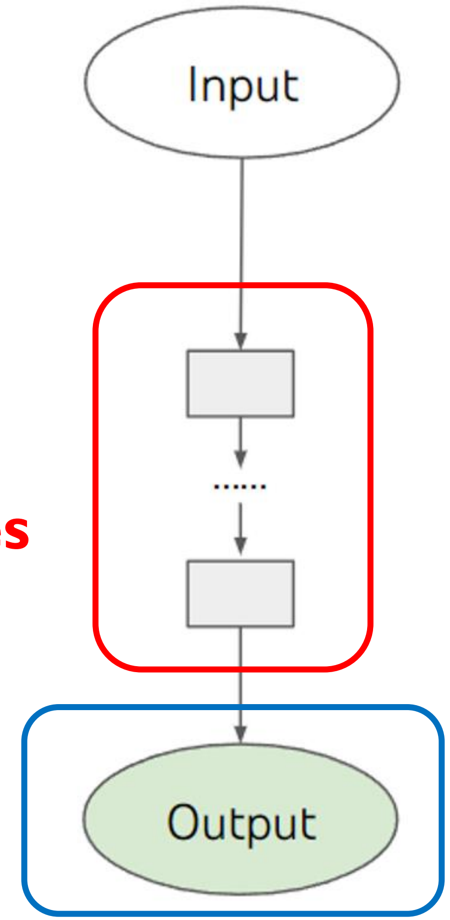
Scaling Test-Time Compute via Verifiers

- So what are verifiers?
  - ORM: Outcome-supervised Reward Model
  - PRM: Process-supervised Reward Model

Next question: How to train a PRM?

PRM <= Output + Label + supervision of rationales

ORM <= Output + Label



A CoT rationale

Scaling Test-Time Compute via Verifiers

- How to train a PRM?

  (We only discuss the case that you cannot afford the annotations by human. 🤣)

Scaling Test-Time Compute via Verifiers

- How to train a PRM?

  (We only discuss the case that you cannot afford the annotations by human. 🤣)

  - Instead of directly annotating each reasoning step, we estimate the quality of them.

Scaling Test-Time Compute via Verifiers

- How to train a PRM?

  (We only discuss the case that you cannot afford the annotations by human. 🤣)

  - Instead of directly annotating each reasoning step, we estimate the quality of them.
  - The quality of a reasoning step is defined as its potential to deduce the correct answer.[4] (Just like a soft label)

Scaling Test-Time Compute via Verifiers

- How to train a PRM?

  (We only discuss the case that you cannot afford the annotations by human. 🤣)

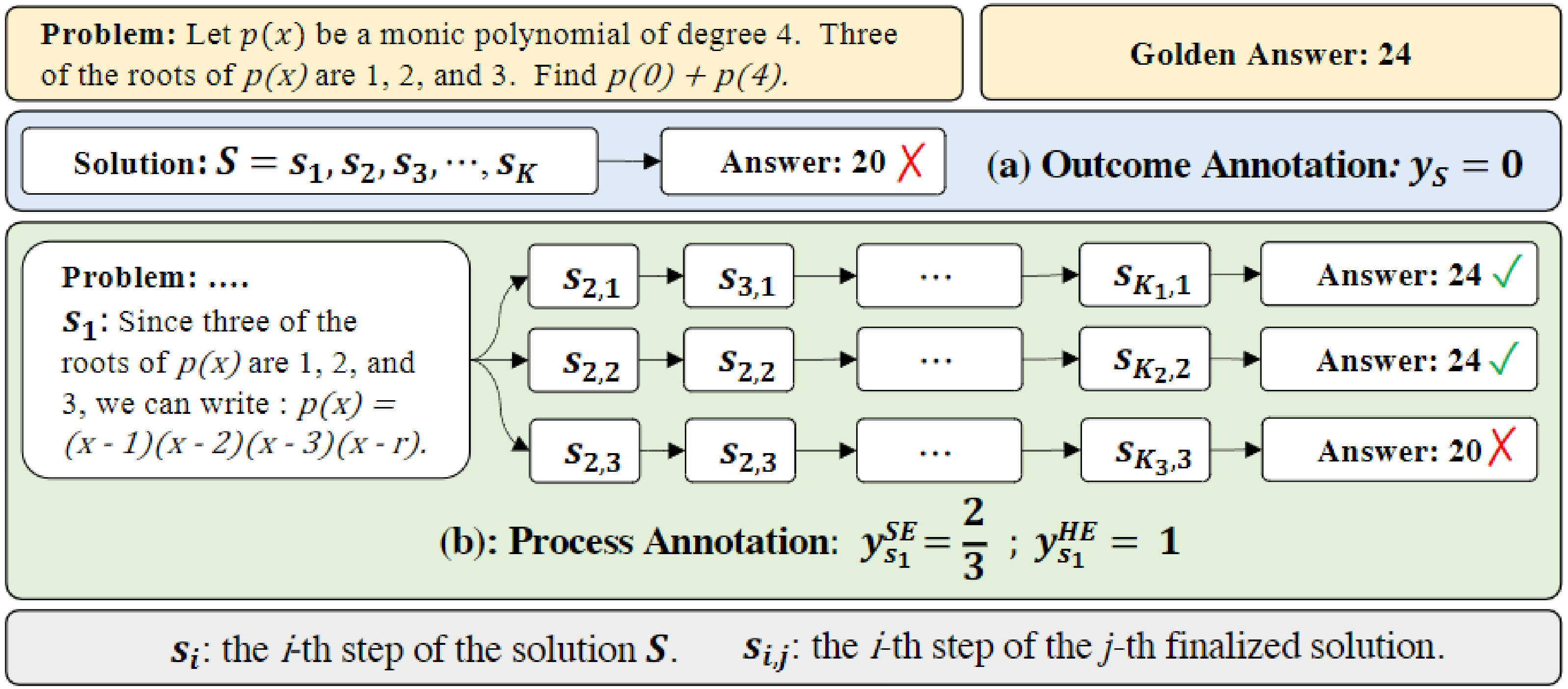
  - Instead of directly annotating each reasoning step, we estimate the quality of them.
  - The quality of a reasoning step is defined as its potential to deduce the correct answer.[4] (Just like a soft label)

[4] Wang et al., "Math-Shepherd: Verify and Reinforce LLMs Step-by-step without Human Annotations" arXiv 24.02

Scaling Test-Time Compute via Verifiers

- How to train a PRM?

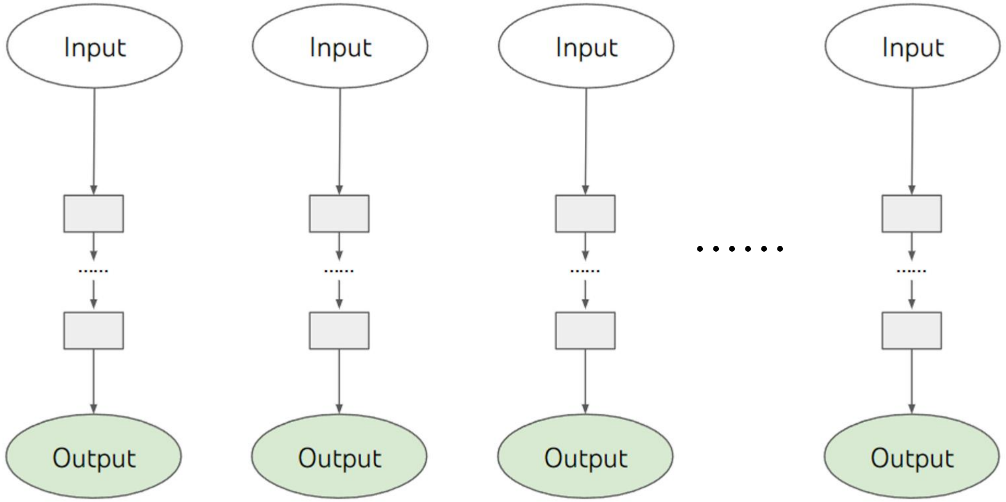
  (We only discuss the case that you cannot afford the annotations by human. 🤣)
  - Instead of directly annotating each reasoning step, we estimate the quality of them.
  - The quality of a reasoning step is defined as its potential to deduce the correct answer.[4] (Just like a soft label)



[4] Wang et al., "Math-Shepherd: Verify and Reinforce LLMs Step-by-step without Human Annotations" arXiv 24.02

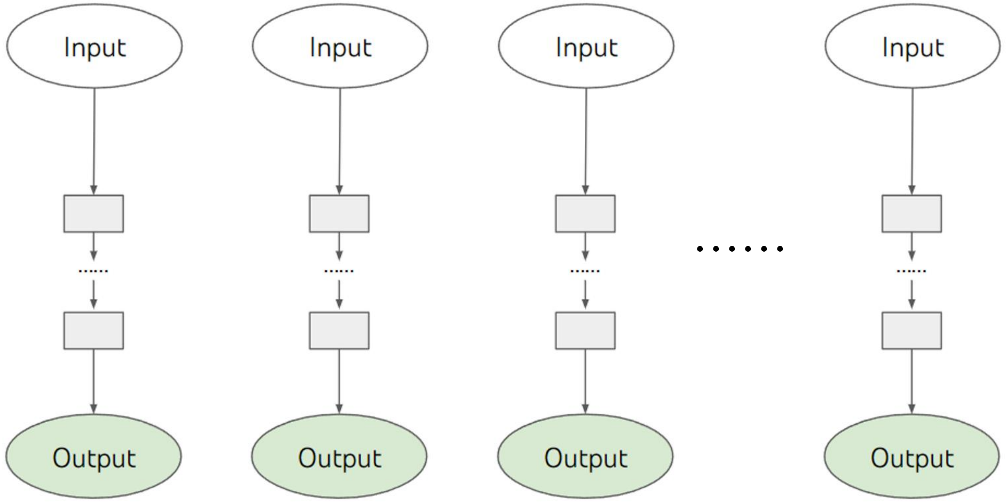
Scaling Test-Time Compute via Verifiers

- How to score with the verifier (Answer aggregation)



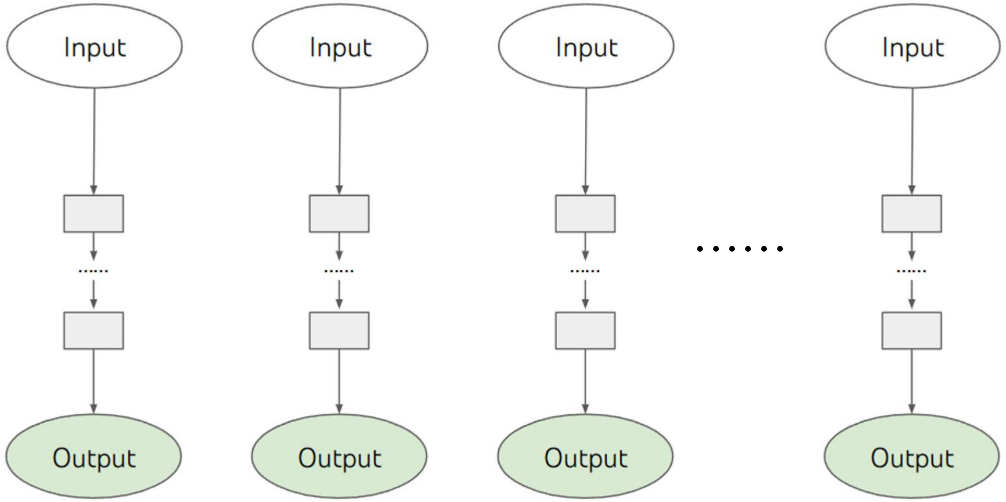
Scaling Test-Time Compute via Verifiers

- How to score with the verifier (Answer aggregation)
  - To select the best-of-N answers with the PRM, we need to aggregate across all the per-step scores for each answer to determine the best candidate.



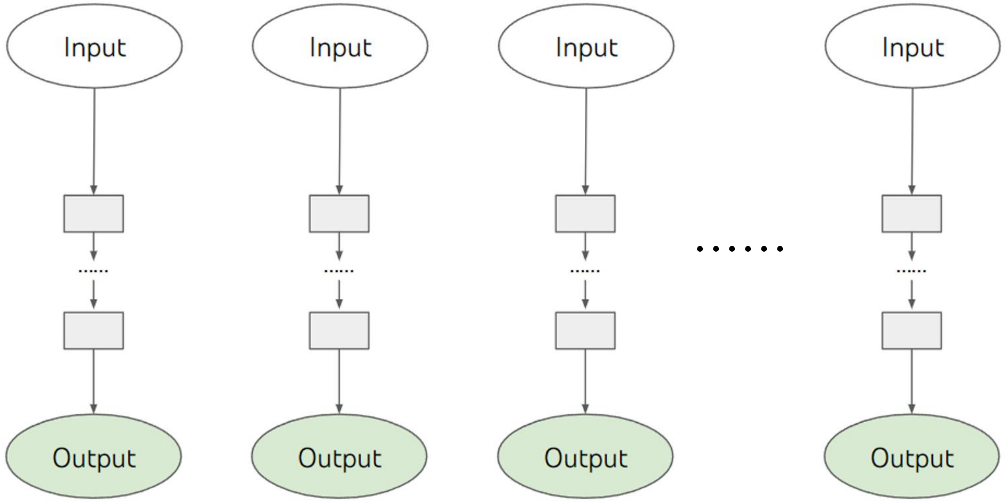
Scaling Test-Time Compute via Verifiers

- How to score with the verifier (Answer aggregation)
  - To select the best-of-N answers with the PRM, we need to aggregate across all the per-step scores for each answer to determine the best candidate.
    - Step-wise aggregation (inside-answer)



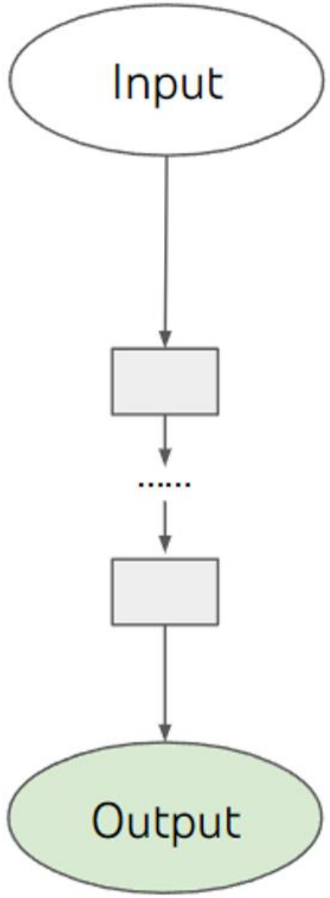
Scaling Test-Time Compute via Verifiers

- How to score with the verifier (Answer aggregation)
  - To select the best-of-N answers with the PRM, we need to aggregate across all the per-step scores for each answer to determine the best candidate.
    - Step-wise aggregation (inside-answer)
    - Inter-answer aggregation (between-answer)



Scaling Test-Time Compute via Verifiers

- How to score with the verifier (Answer aggregation)

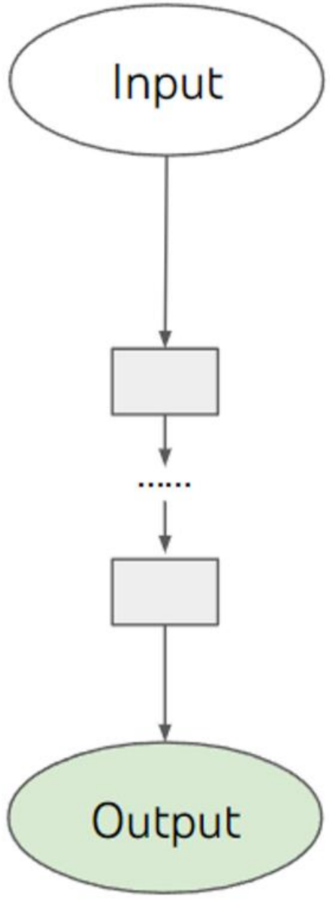


A CoT rationale

[5] Lightman et al. (OpenAI), "Let's verify step by step" ICLR 2024

Scaling Test-Time Compute via Verifiers

- How to score with the verifier (Answer aggregation)
  - Step-wise aggregation

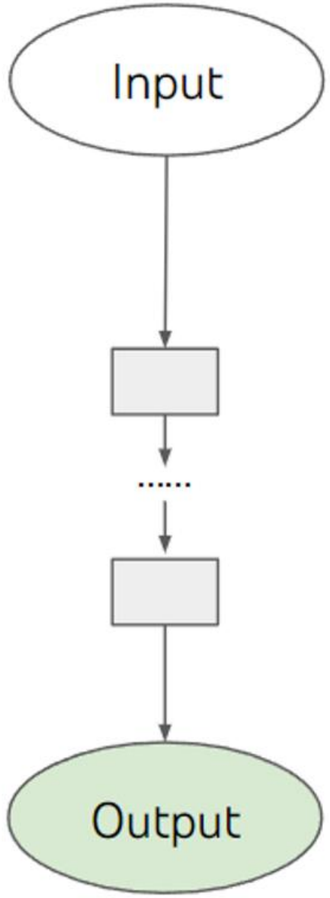


A CoT rationale

[5] Lightman et al. (OpenAI), "Let's verify step by step" ICLR 2024

Scaling Test-Time Compute via Verifiers

- How to score with the verifier (Answer aggregation)
  - Step-wise aggregation
    - (How to calculate the score for a single answer?)

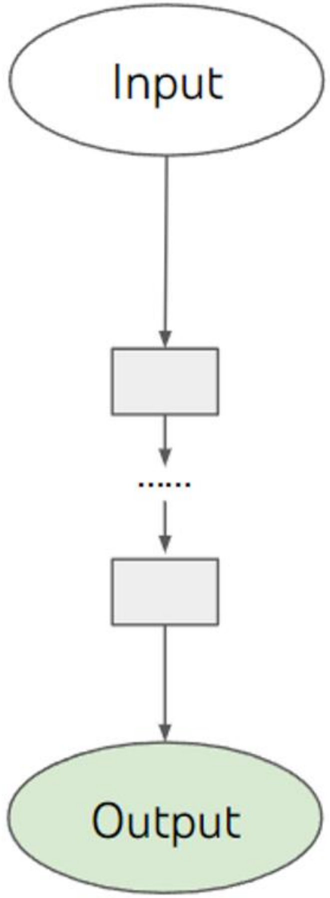


A CoT rationale

[5] Lightman et al. (OpenAI), "Let's verify step by step" ICLR 2024

Scaling Test-Time Compute via Verifiers

- How to score with the verifier (Answer aggregation)
  - Step-wise aggregation
    - (How to calculate the score for a single answer?)
    - Some work[4][5] aggregating the per-step scores by taking the product or minimum

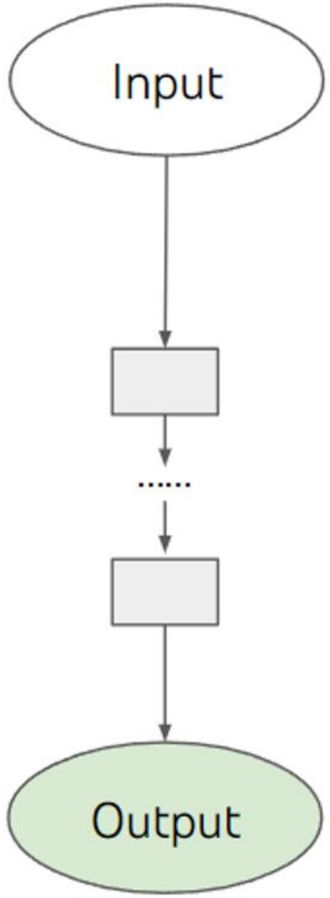


A CoT rationale

[5] Lightman et al. (OpenAI), "Let's verify step by step" ICLR 2024

Scaling Test-Time Compute via Verifiers

- How to score with the verifier (Answer aggregation)
  - Step-wise aggregation
    - (How to calculate the score for a single answer?)
    - Some work[4][5] aggregating the per-step scores by taking the product or minimum
    - This paper finds that using the score of the last step performs best with their PRM.

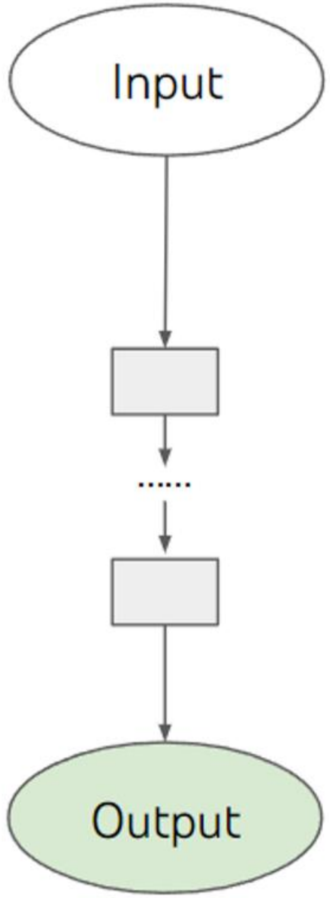


A CoT rationale

[5] Lightman et al. (OpenAI), "Let's verify step by step" ICLR 2024

Scaling Test-Time Compute via Verifiers

- How to score with the verifier (Answer aggregation)
  - Step-wise aggregation
    - (How to calculate the score for a single answer?)
    - Some work[4][5] aggregating the per-step scores by taking the product or minimum
    - This paper finds that using the score of the last step performs best with their PRM.
  - Inter-answer aggregation

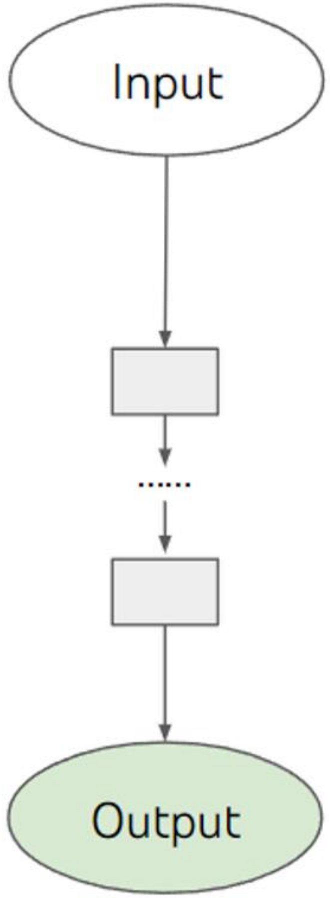


A CoT rationale

[5] Lightman et al. (OpenAI), "Let's verify step by step" ICLR 2024

Scaling Test-Time Compute via Verifiers

- How to score with the verifier (Answer aggregation)
  - Step-wise aggregation
    - (How to calculate the score for a single answer?)
    - Some work[4][5] aggregating the per-step scores by taking the product or minimum
    - This paper finds that using the score of the last step performs best with their PRM.
  - Inter-answer aggregation
    - (How to choose the best answer candidate)

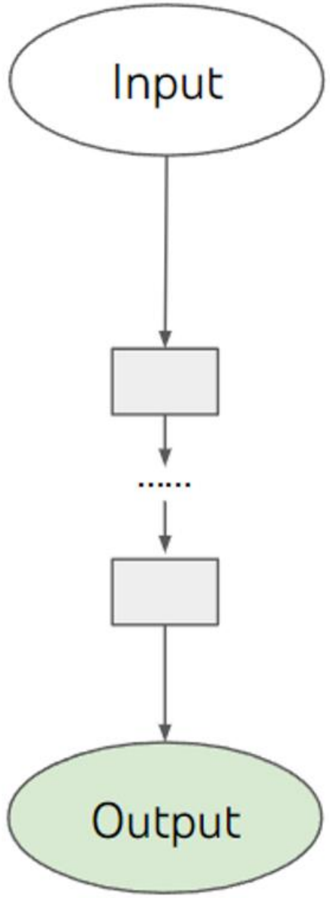


A CoT rationale

[5] Lightman et al. (OpenAI), "Let's verify step by step" ICLR 2024

Scaling Test-Time Compute via Verifiers

- How to score with the verifier (Answer aggregation)
  - Step-wise aggregation
    - (How to calculate the score for a single answer?)
    - Some work[4][5] aggregating the per-step scores by taking the product or minimum
    - This paper finds that using the score of the last step performs best with their PRM.
  - Inter-answer aggregation
    - (How to choose the best answer candidate)
    - Marginalizing scores across all solutions with the same final answer. (“weighted aggregation”)

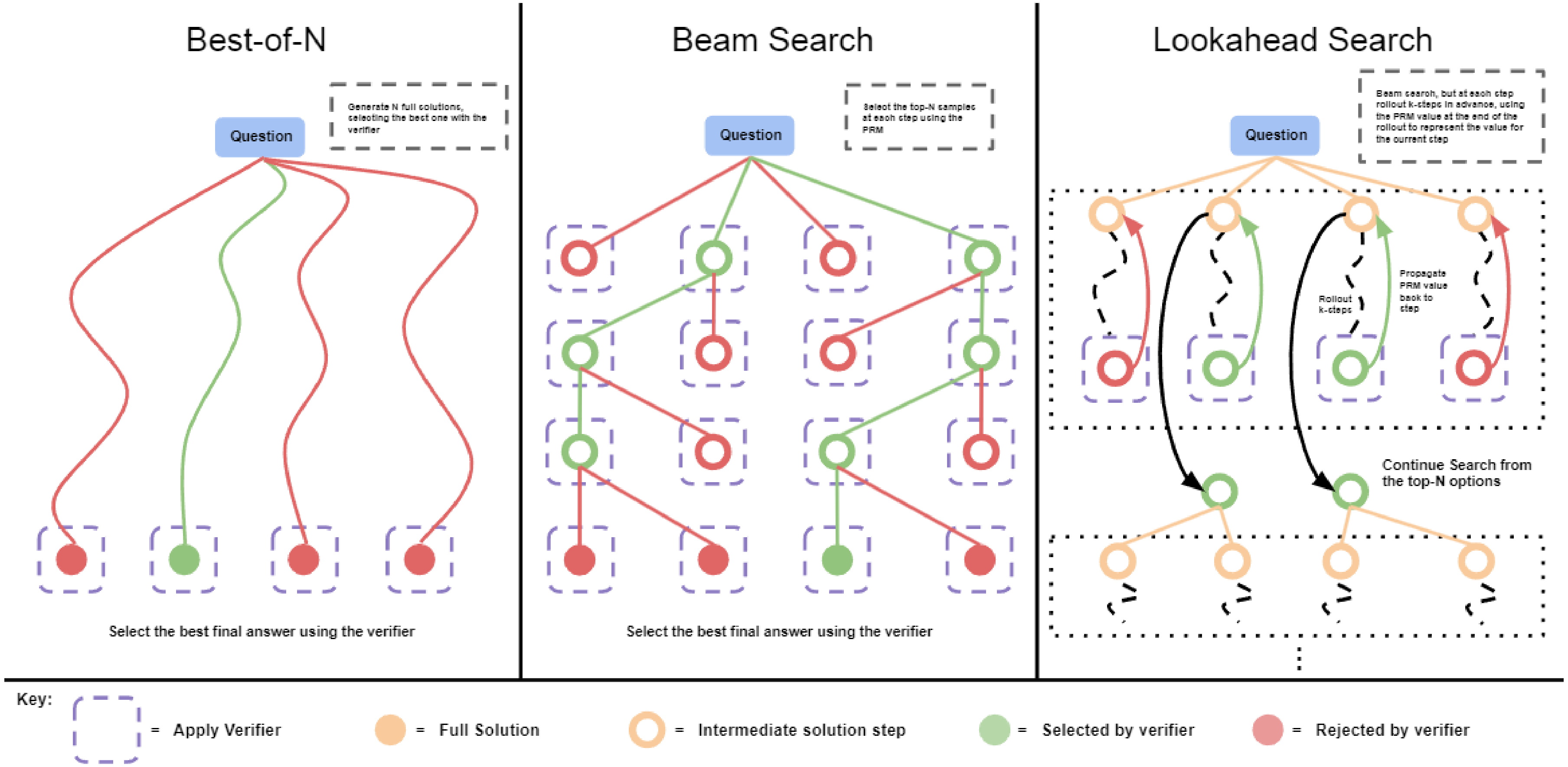


A CoT rationale

[5] Lightman et al. (OpenAI), “Let’s verify step by step” ICLR 2024

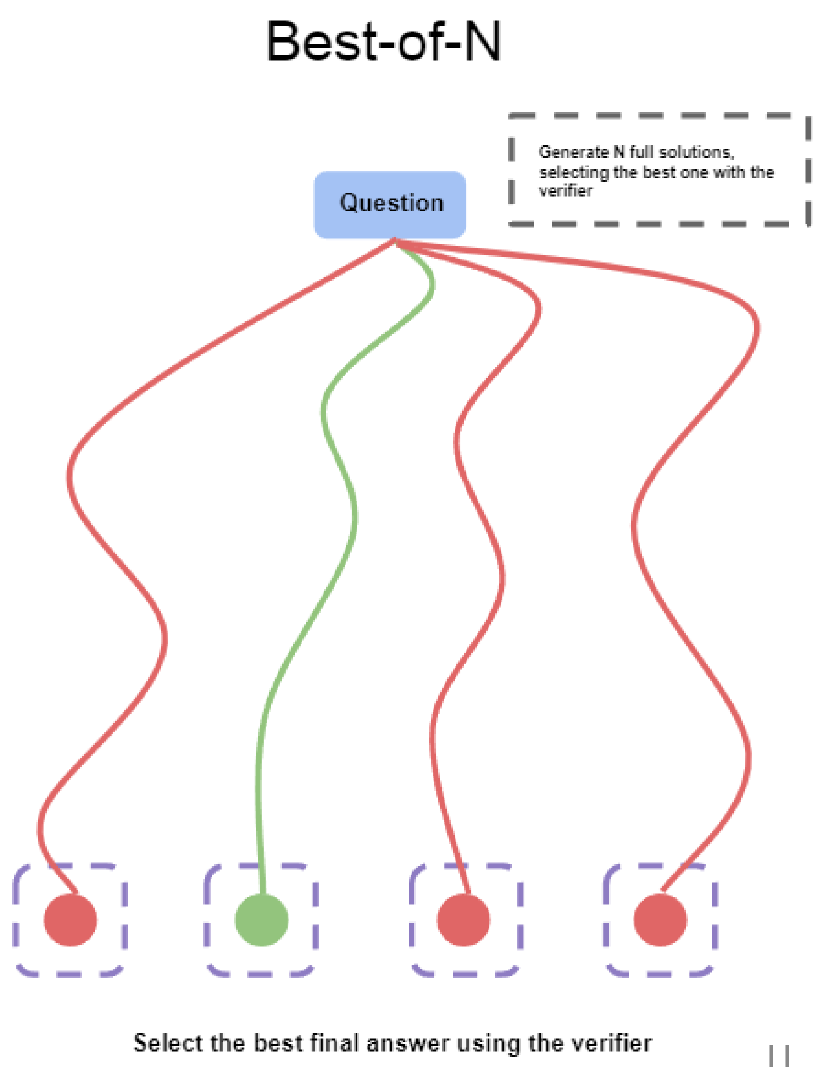
Scaling Test-Time Compute via Verifiers

- Search Methods Against a verifier



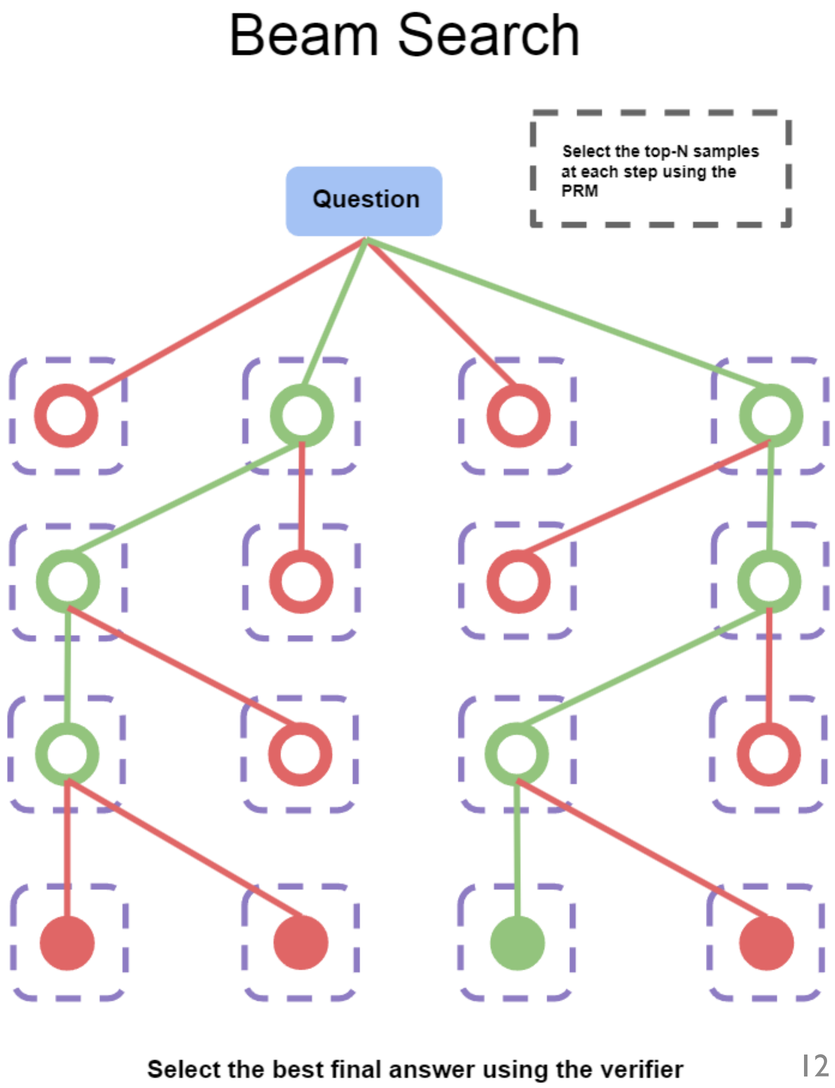
Scaling Test-Time Compute via Verifiers

- Search Methods Against a verifier
  - (weighted) Best-of-N
  - Just sample N answers independently from the base LLM
  - Select the candidate according to the PRM's answer aggregation calculation.



Scaling Test-Time Compute via Verifiers

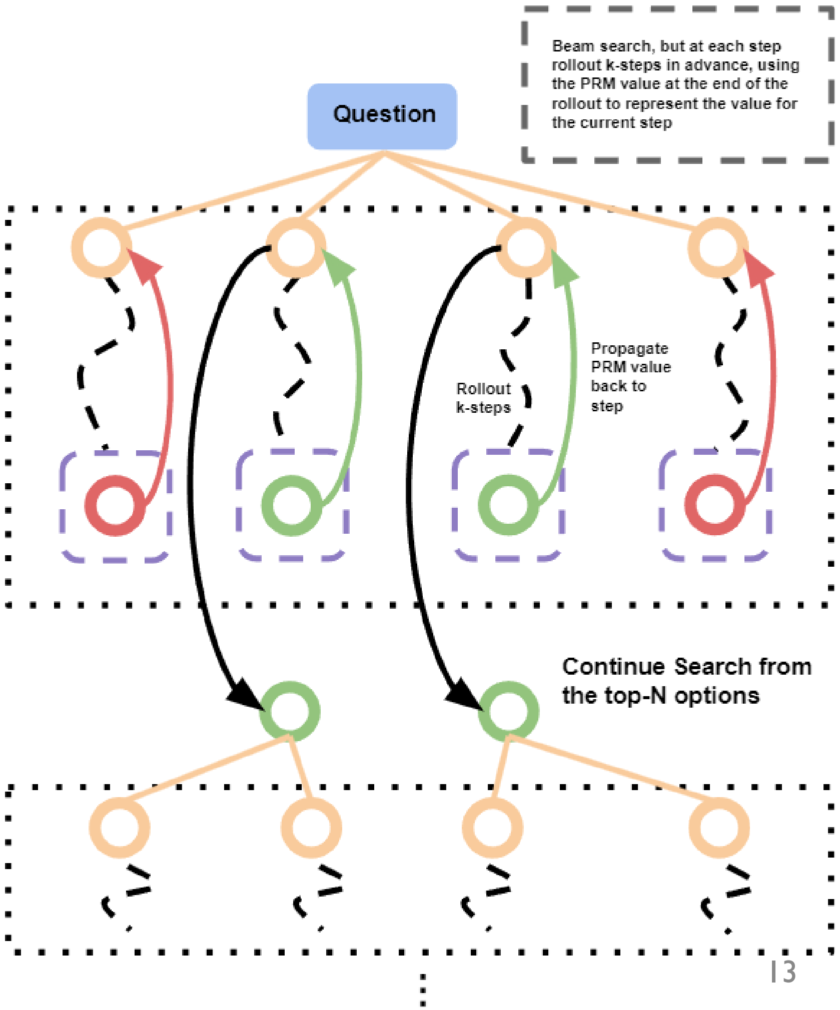
- Search Methods Against a verifier
  - Beam Search
  - Control a total number N and a beam width M (N=4, M=2)
  - Similar to the to the LM decoding strategy “beam search” (Difference that each node denotes the intermediate reasoning step here.)



Scaling Test-Time Compute via Verifiers

- Search Methods Against a verifier
  - Lookahead Search
  - Based on beam search, it modifies how to evaluate each step.
  - Rollout k steps and having the score at the k-th step as the score of current reasoning rationale.
  - (Main idea is just like A* / Monte-Carlo Tree Search)

Lookahead Search

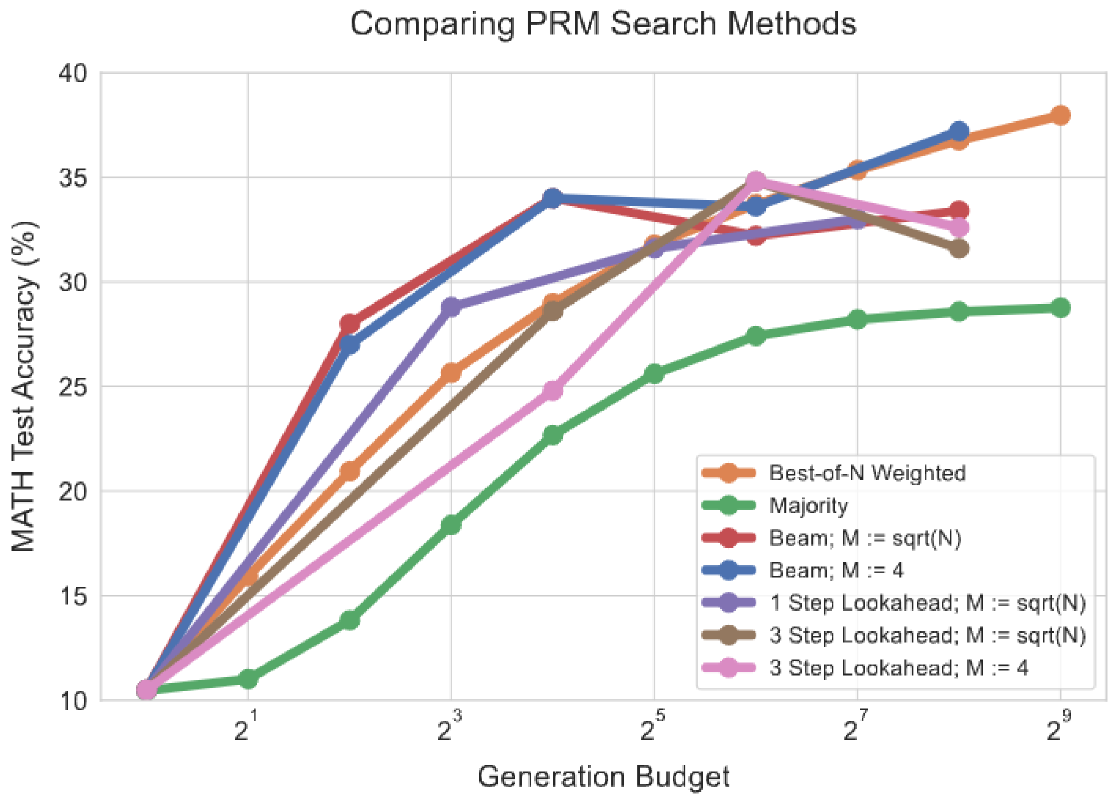


Scaling Test-Time Compute via Verifiers

- Experimental setup
  - Two main factors affecting the performances
  - Generation budget
    - e.g. Number of sampling
  - Difficulty of question
    - Easy questions may do not require much reasoning, while hard questions need much reasoning.

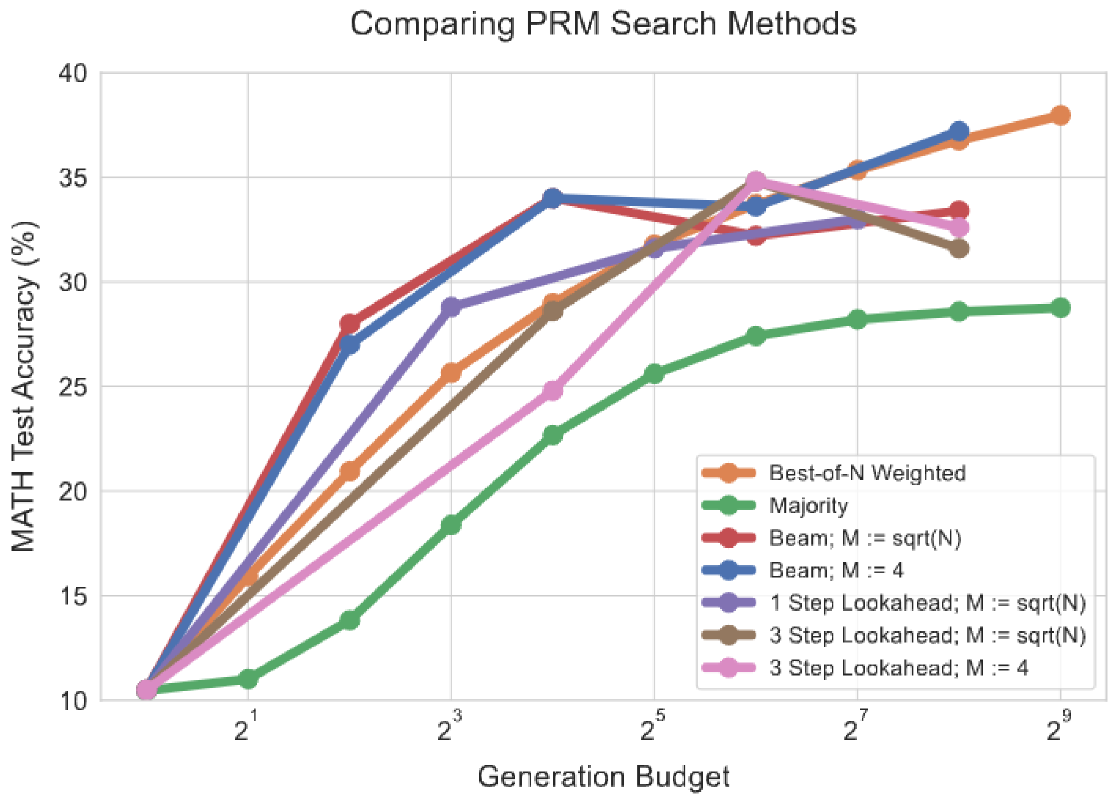
Scaling Test-Time Compute via Verifiers

- Results & Findings
  - When budget is small,
    beam search > best-of-N > lookahead
  - When budget is large,
    best-of-N > beam search > lookahead



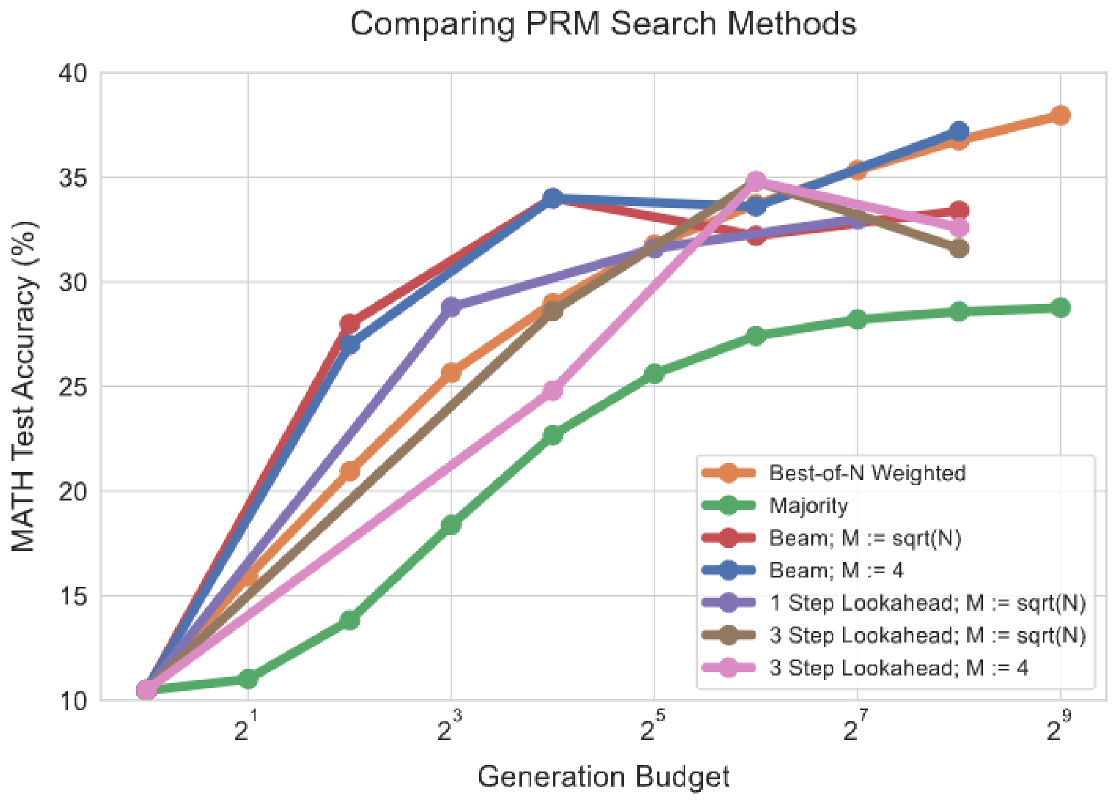
Scaling Test-Time Compute via Verifiers

- Results & Findings
  - When budget is small,
    beam search > best-of-N > lookahead
  - When budget is large,
    best-of-N > beam search > lookahead
- Possible explanations



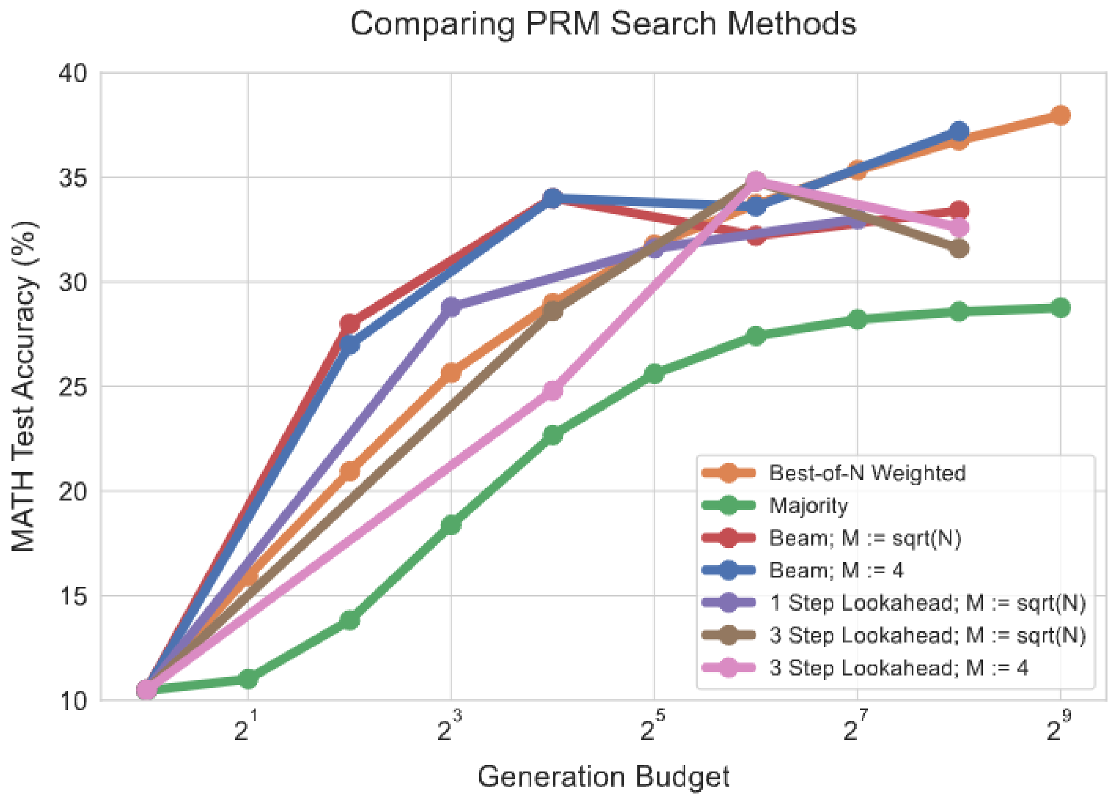
Scaling Test-Time Compute via Verifiers

- Results & Findings
  - When budget is small,
    beam search > best-of-N > lookahead
  - When budget is large,
    best-of-N > beam search > lookahead
- Possible explanations
  - When budget is small, we need more sophisticated searching strategy (simply sampling may be hard to hit).



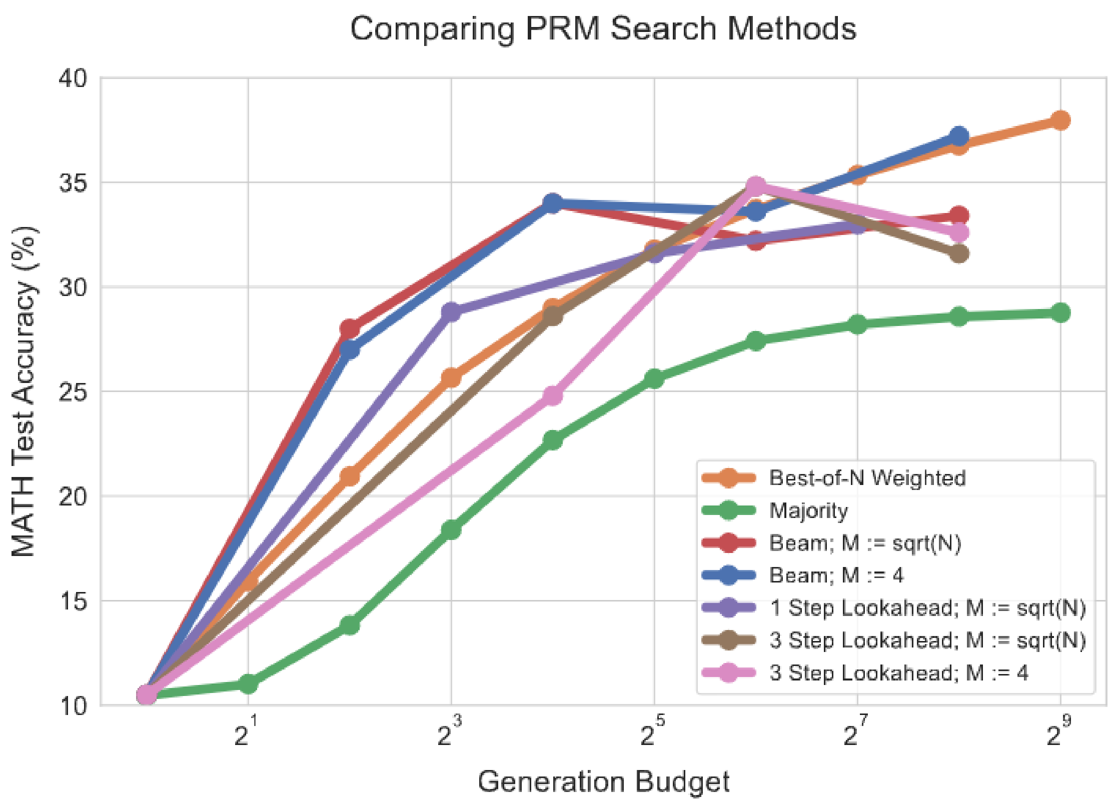
Scaling Test-Time Compute via Verifiers

- Results & Findings
  - When budget is small,
    beam search > best-of-N > lookahead
  - When budget is large,
    best-of-N > beam search > lookahead
- Possible explanations
  - When budget is small, we need more sophisticated searching strategy (simply sampling may be hard to hit).
  - When budget is large, it will alleviate this problem.



Scaling Test-Time Compute via Verifiers

- Results & Findings
  - When budget is small,
    beam search > best-of-N > lookahead
  - When budget is large,
    best-of-N > beam search > lookahead
- Possible explanations
  - When budget is small, we need more sophisticated searching strategy (simply sampling may be hard to hit).
  - When budget is large, it will alleviate this problem.
  - Lookahead search generally underperforms, probably due to over-optimizing for searching.

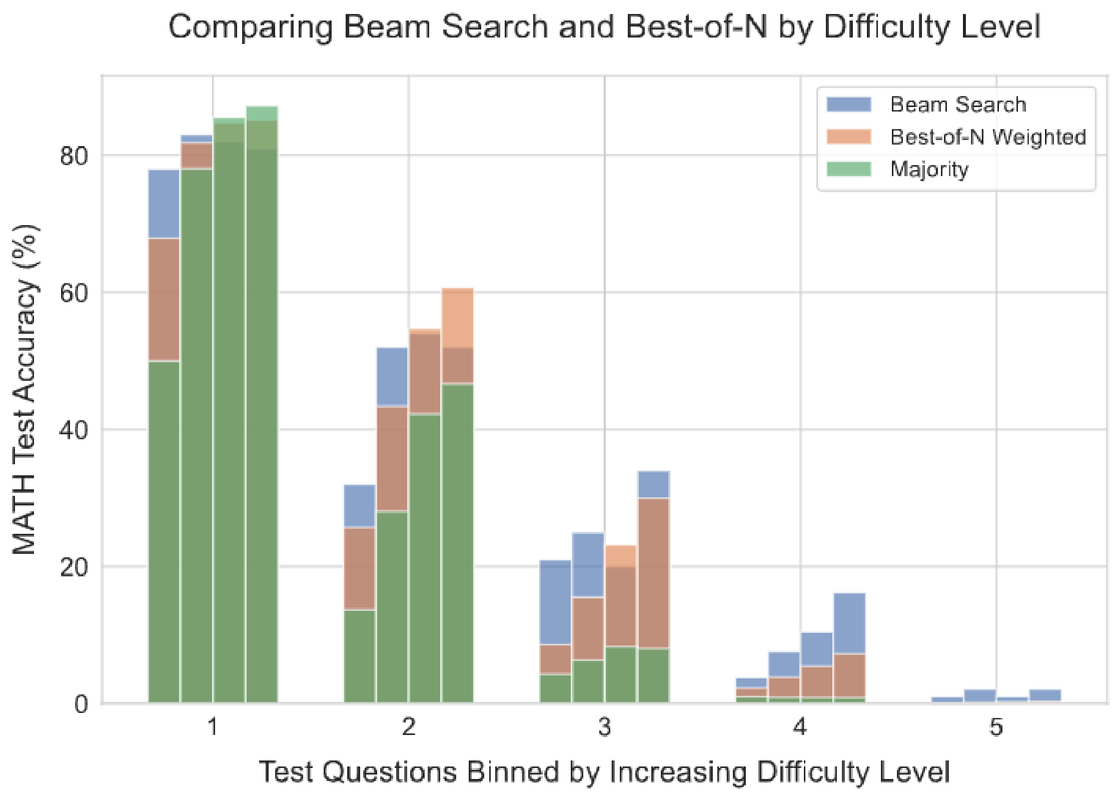


Scaling Test-Time Compute via Verifiers

- Results & Findings

  (Four Bars under each bin demotes the searching budget: 4, 16, 64, 256)

  - On simplest questions (1, 2), best-of-N > beam search
  - On hard questions (3, 4), beam search > best-of-N
  - On the hardest questions (5), nothing works



Refining the Proposal Distribution

- LLMs can learn to generate better rationales via SFT with high-quality data.
- After that, we can let LLMs refine their proposal distributions.
- Primarily there are two strategies for refining output distribution
  - Parallel sampling
  - Sequential revisions

If $f(x) = \frac{3x-2}{x-2}$, what is the value of $f(-2) +f(-1)+f(0)$? Express your answer as a common fraction.

Attempt 1:

```
Let's evaluate $f(-2)$.
We have $f(-2)=\frac{3(-2)-2}{-2-2}=-\frac{4}{-4}=\frac{1}{1}=1$.
So $f(-2)=1$.
Let's evaluate $f(-1)$.
We have $f(-1)=\frac{3(-1)-2}{-1-2}=-\frac{5}{-3}=\frac{5}{3}$.
So $f(-1)=\frac{5}{3}$.
Let's evaluate $f(0)$.
We have $f(0)=\frac{3(0)-2}{0-2}=\frac{-2}{-2}=1$.
So $f(0)=1$.
So we want to compute
$1+\frac{5}{3}+1=\frac{3}{3}+\frac{5}{3}=\frac{3+5}{3}=\frac{8}{3}$.
So the answer is $\boxed{\frac{8}{3}}$.
#### \frac{8}{3}
```

Attempt 2:

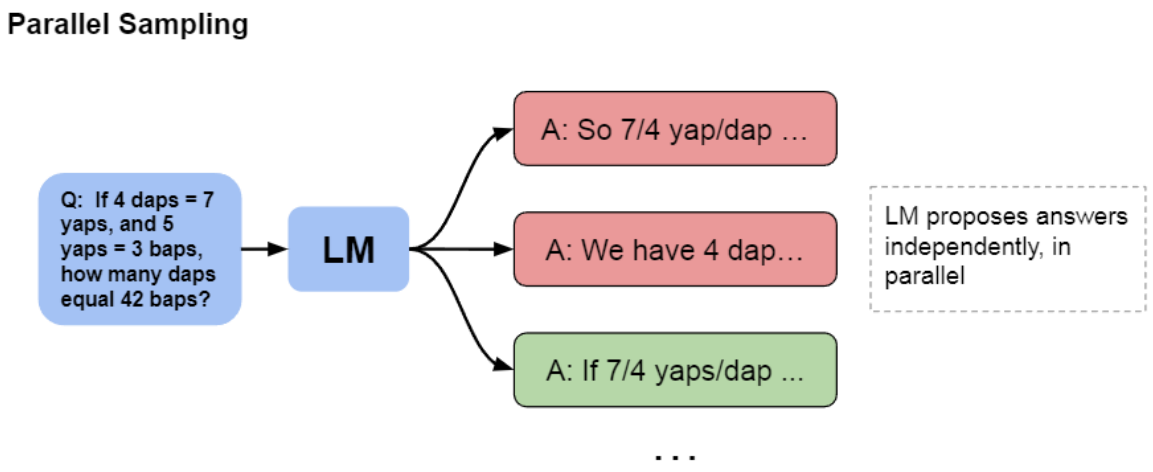
```
Let's first find $f(-2)$.
$f(-2) = \frac{3(-2)-2}{-2-2} = \frac{-6-2}{-4} = \frac{-8}{-4} = 2$.
Now let's find $f(-1)$.
$f(-1) = \frac{3(-1)-2}{-1-2} = \frac{-3-2}{-3} = \frac{-5}{-3} = \frac{5}{3}$.
Now let's find $f(0)$.
$f(0) = \frac{3(0)-2}{0-2} = \frac{-2}{-2} = 1$.
So we have $f(-2) +f(-1)+f(0) = 2+\frac{5}{3}+1 =
\frac{7}{3}+\frac{5}{3}+\frac{3}{3} = \frac{7+5+3}{3} = \frac{15}{3} = 5$.
So the answer is $\boxed{5}$.
#### 5
```

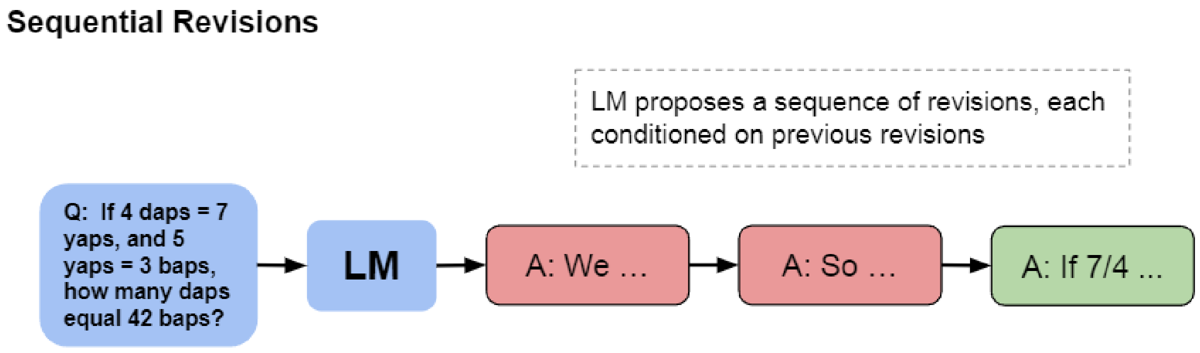
Attempt 3:

```
Let's plug in $-2$, $-1$, and $0$ for $x$ in $f(x)$.
We get $f(-2) = \frac{3(-2)-2}{-2-2} = \frac{-8}{-4} = \frac{2}{1} = 2, f(-1) =
\frac{3(-1)-2}{-1-2} = \frac{-5}{-3} = \frac{5}{3}, f(0) = \frac{3(0)-2}{0-2} =
\frac{-2}{-2} = 1$.
So we have $2+\frac{5}{3}+1=\frac{6}{3}+\frac{5}{3}+\frac{3}{3}=\frac{14}{3}$.
So the answer is $\boxed{\frac{14}{3}}$.
#### \frac{14}{3}
```

Refining the Proposal Distribution

- Two major methods for refining the proposal distribution
  - **Parallel Sampling** v.s. **Sequential Revisions**
  - (global search v.s. local refinement)





Refining the Proposal Distribution

- However, there are many problems

Refining the Proposal Distribution

- However, there are many problems
  - E.g.

Refining the Proposal Distribution

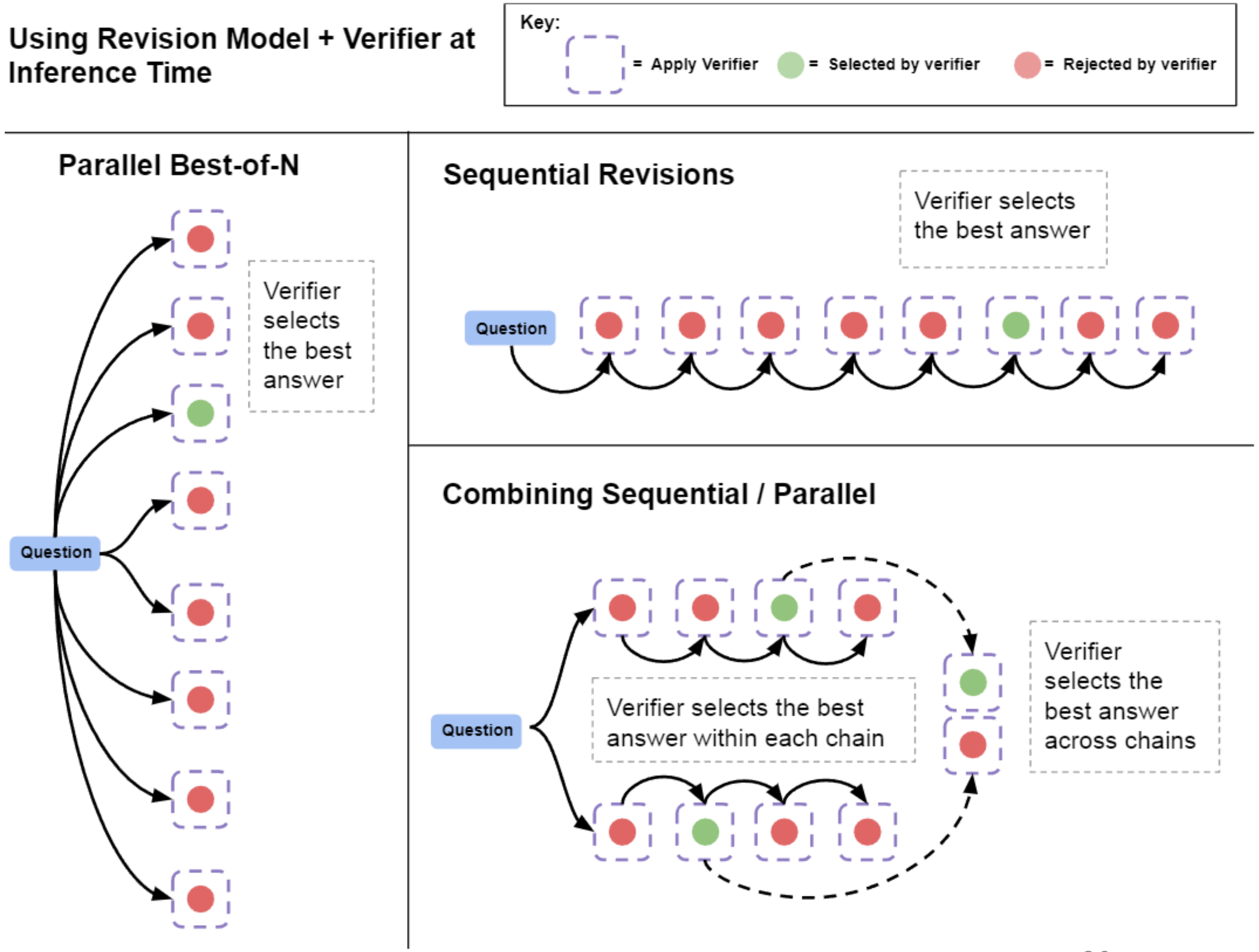
- However, there are many problems
  - E.g.
  - For sequential revision, the last attempt is not guaranteed to be correct. (There is case that it is revised correctly in the middle, and then revised incorrectly at last.)

Refining the Proposal Distribution

- However, there are many problems
  - E.g.
  - For sequential revision, the last attempt is not guaranteed to be correct. (There is case that it is revised correctly in the middle, and then revised incorrectly at last.)
  - For both of them, it’s not guaranteed to have correct attempts.

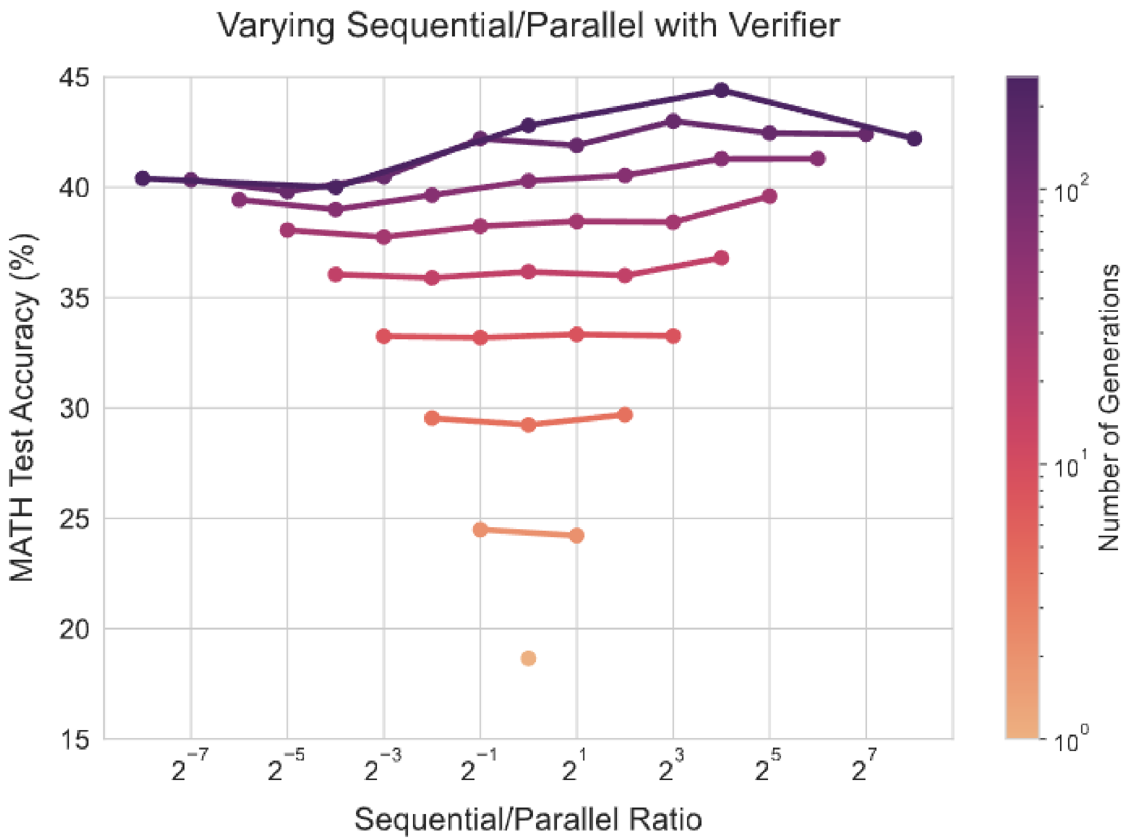
Refining the Proposal Distribution

- Utilizing verifiers to help refinement
  - Parallel Best-of-N
  - Sequential Revisions
  - Combining Sequential / Parallel
    - Trading off between them?



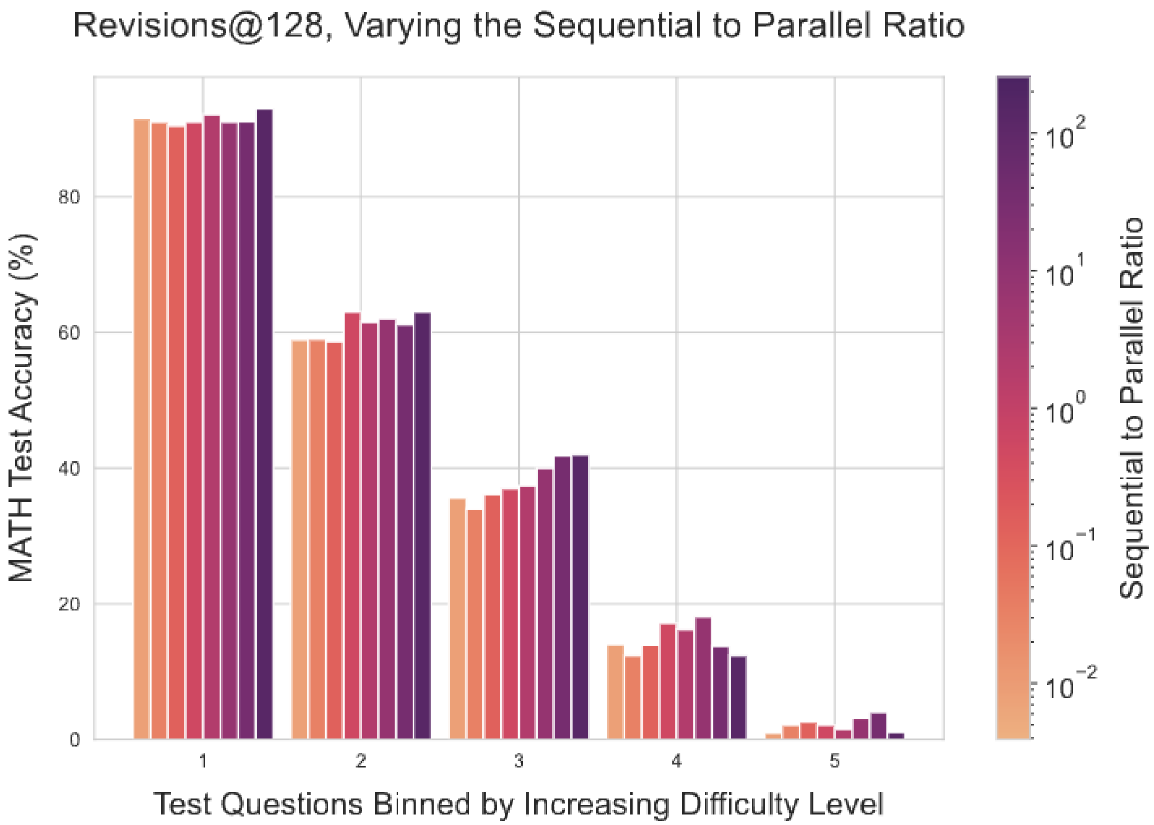
Refining the Proposal Distribution

- Trading off between parallel sampling & sequential revisions
  - (Generation budget)
  - Under low budget, performances increase with more sequential revisions.
  - Under higher budgets, there is an ideal ratio that strikes a balance between them.



Refining the Proposal Distribution

- Trading off between parallel sampling & sequential revisions
  - (Question difficulty)
  - Easier questions attain the best performance with full sequential compute.
  - On the harder questions, there is an ideal ratio of sequential to parallel test-time compute.



Pre-train or Inference?

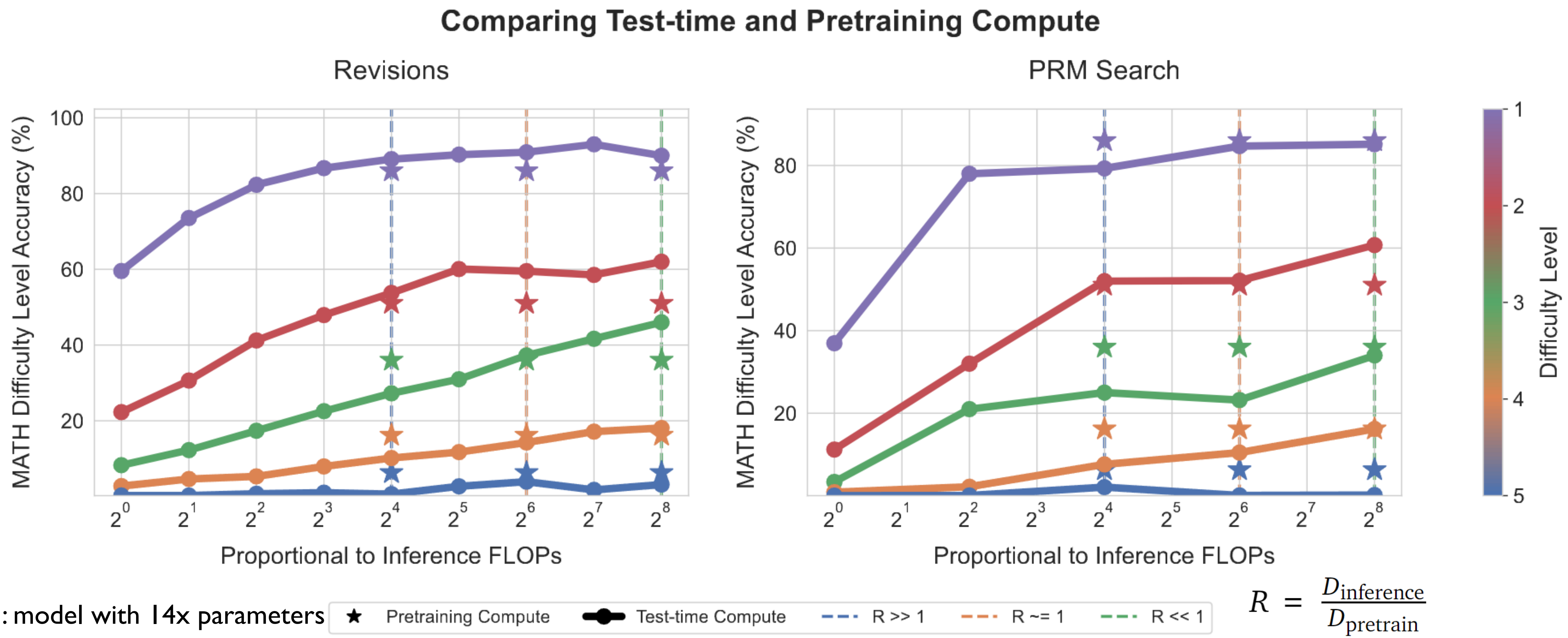
- Q: How much better can the results under the inference scaling law be than under the pretraining scaling law?

Pre-train or Inference?

- Q: How much better can the results under the inference scaling law be than under the pretraining scaling law?
- In other words, if we assign the same amount of computing to inference and pretrain, how about the performances?

Pre-train or Inference?

- Experimental results

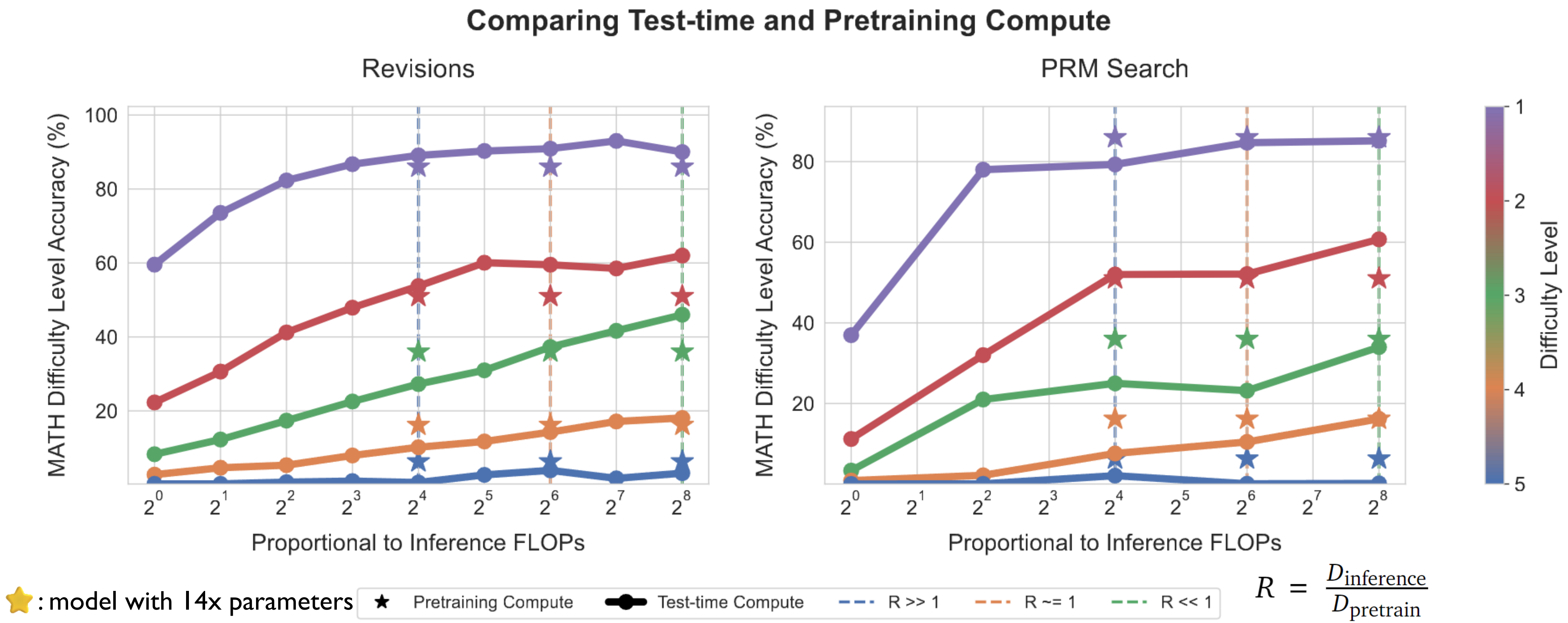


⭐: model with 14x parameters

$$R = \frac{D_{\text{inference}}}{D_{\text{pretrain}}}$$

Findings

1. For easy questions or in settings with a lower inference load ($R << 1$), test-time compute can generally outperform scaling model parameters.
2. For harder questions or in settings with a higher inference load ($R >> 1$), pretraining is a more effective way to improve performance.



$$R = \frac{D_{\text{inference}}}{D_{\text{pretrain}}}$$

Takeaways for exchanging pretrain and test-time compute

Takeaways for exchanging pretrain and test-time compute

- Test-time and pretraining compute are **not** 1-to-1 “exchangeable”.

Takeaways for exchanging pretrain and test-time compute

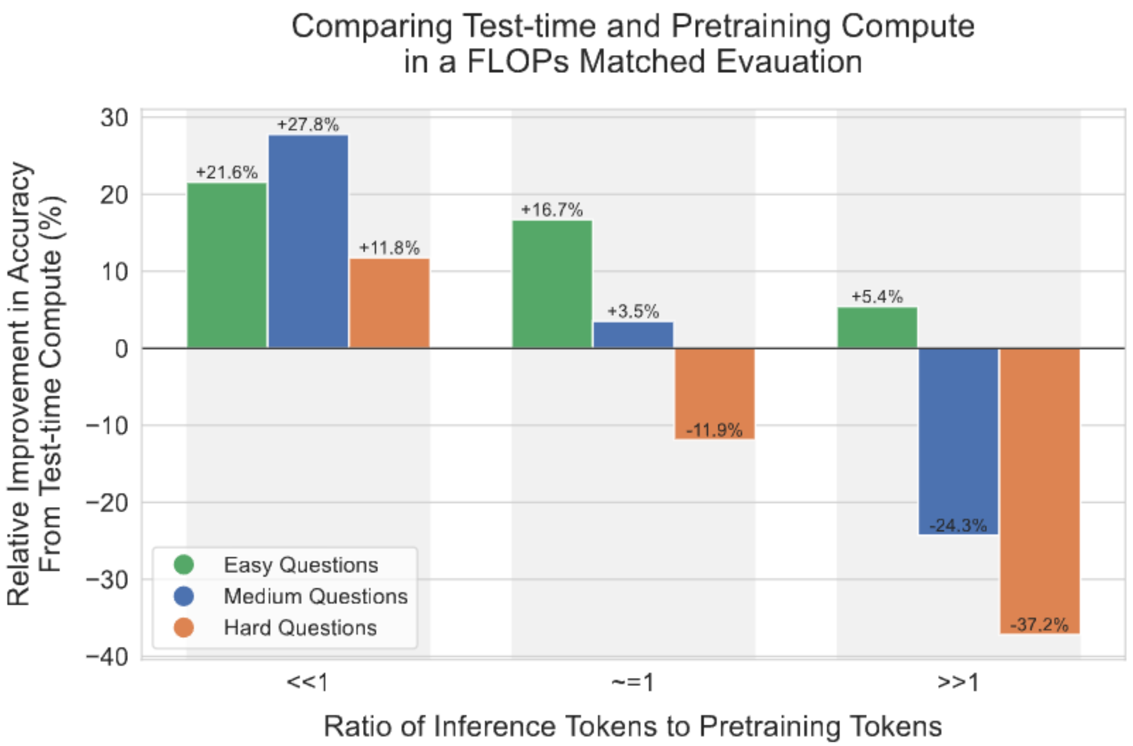
- Test-time and pretraining compute are not 1-to-1 "exchangeable".
- On easy and medium questions, which are within a model's capabilities, or in settings with small inference requirement, test-time compute can easily cover up for additional pretraining.

Takeaways for exchanging pretrain and test-time compute

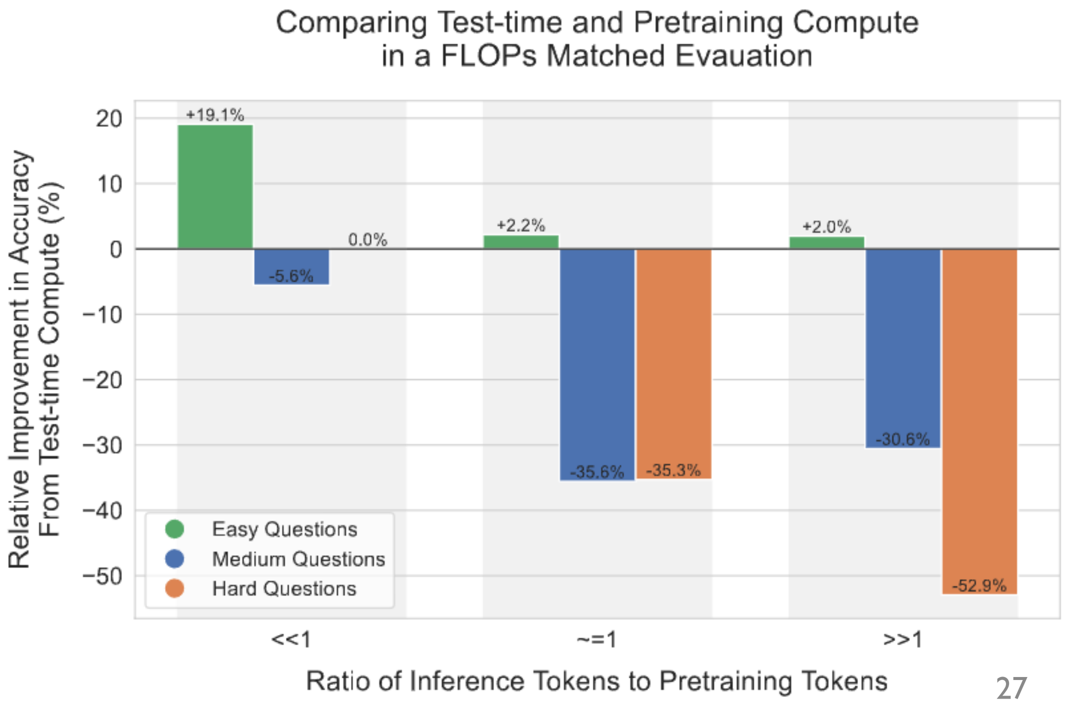
- Test-time and pretraining compute are not 1-to-1 “exchangeable”.
- On easy and medium questions, which are within a model’s capabilities, or in settings with small inference requirement, test-time compute can easily cover up for additional pretraining.
- However, on challenging questions which are outside a given base model’s capabilities or under higher inference requirement, pretraining is likely more effective for improving performance.

Takeaways for exchanging pretrain and test-time compute

Iteratively Revising Answers at Test-time



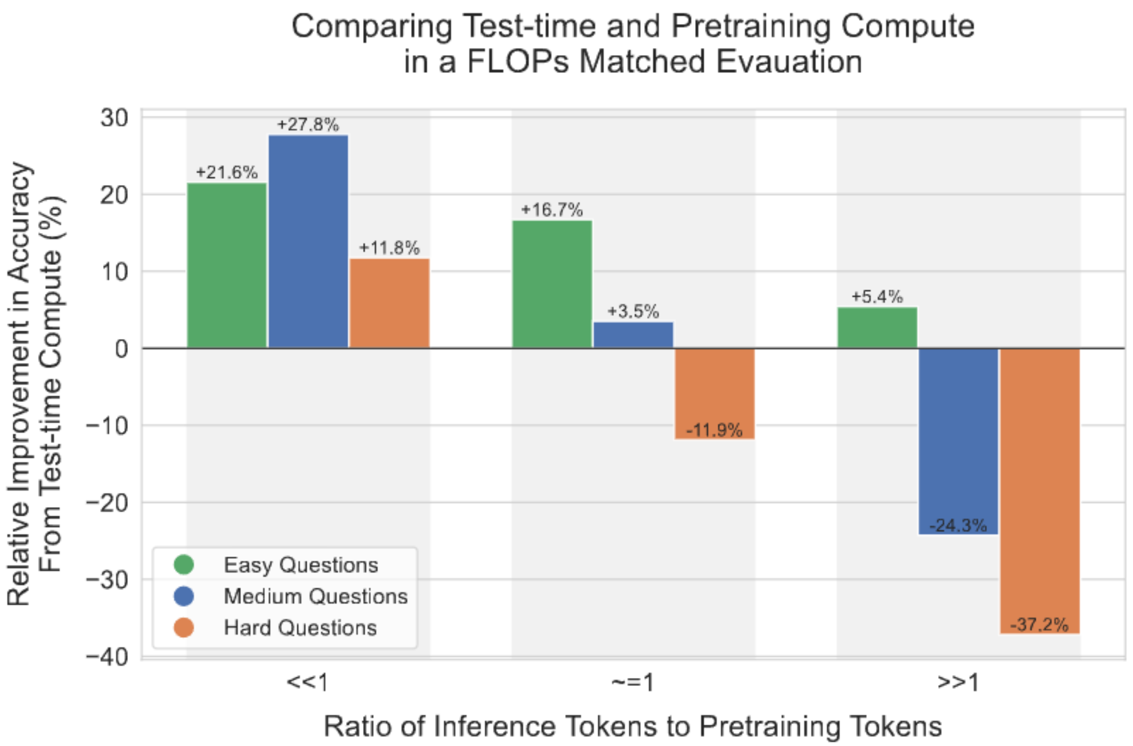
Test-time Search Against a PRM Verifier



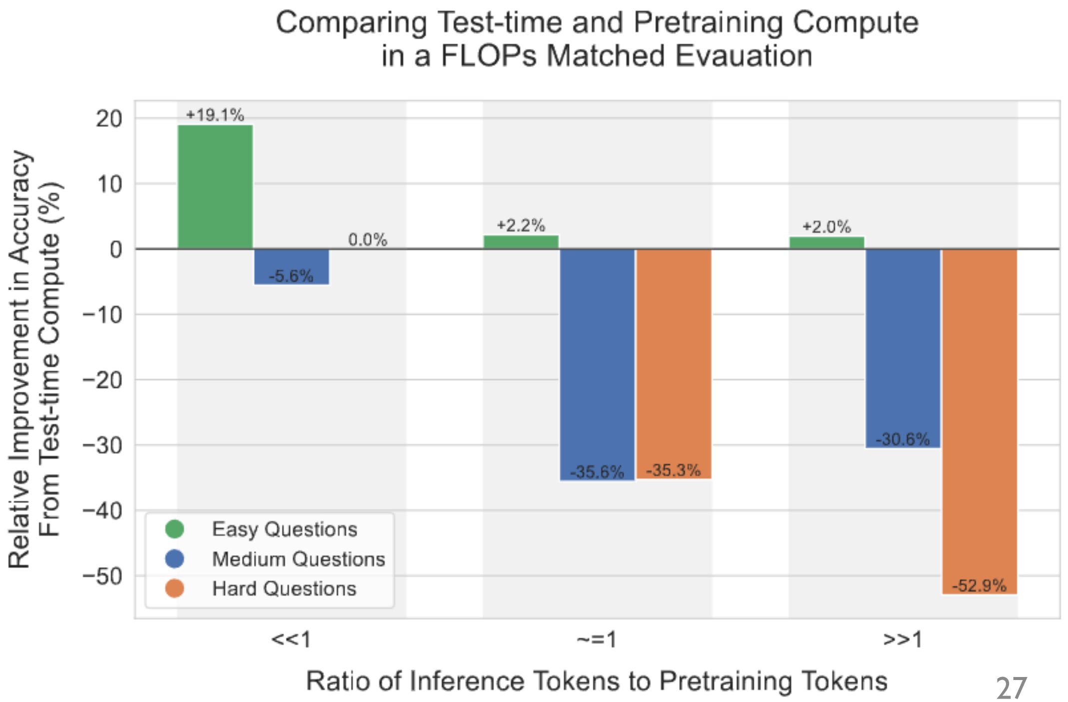
Takeaways for exchanging pretrain and test-time compute

- Some sum-up experimental results

Iteratively Revising Answers at Test-time



Test-time Search Against a PRM Verifier



Take-home messages

- Takeaways
  - For compute-optimal scaling of verifiers

Take-home messages

- Takeaways
  - For compute-optimal scaling of verifiers
  - Beam-search is more effective on harder questions and at lower compute budgets, whereas best-of-N is more effective on easier questions and at higher budgets.

Take-home messages

- Takeaways
  - For compute-optimal scaling of verifiers
  - Beam-search is more effective on harder questions and at lower compute budgets, whereas best-of-N is more effective on easier questions and at higher budgets.
  - Moreover, by selecting the best search setting for a given question difficulty and test-time compute budget, we can nearly outperform best-of-N using up to 4x less test-time compute.

Take-home messages

- Takeaways
  - For compute-optimal scaling by refining the proposal distribution with revisions

Take-home messages

- Takeaways
  - For compute-optimal scaling by refining the proposal distribution with revisions
  - There exists a tradeoff between sequential (e.g. revisions) and parallel (e.g. standard best-of-N) test-time computation, and the ideal ratio of sequential to parallel test-time compute depends on both the compute budget and the specific question at hand.

Take-home messages

- Takeaways
  - For compute-optimal scaling by refining the proposal distribution with revisions
  - There exists a tradeoff between sequential (e.g. revisions) and parallel (e.g. standard best-of-N) test-time computation, and the ideal ratio of sequential to parallel test-time compute depends on both the compute budget and the specific question at hand.
  - Specifically, easier questions benefit from purely sequential test-time compute, whereas harder questions often perform best with some ideal ratio of sequential to parallel compute.

Take-home messages

- Takeaways
  - For compute-optimal scaling by refining the proposal distribution with revisions
  - There exists a tradeoff between sequential (e.g. revisions) and parallel (e.g. standard best-of-N) test-time computation, and the ideal ratio of sequential to parallel test-time compute depends on both the compute budget and the specific question at hand.
  - Specifically, easier questions benefit from purely sequential test-time compute, whereas harder questions often perform best with some ideal ratio of sequential to parallel compute.
  - Moreover, by optimally selecting the best setting for a given question difficulty and test-time compute budget, we can outperform the parallel best-of-N baseline using up to 4x less test-time compute.

Take-home messages

Take-home messages

- Test-time and pretraining compute are not 1-to-1 “exchangeable”.

Take-home messages

- Test-time and pretraining compute are not 1-to-1 “exchangeable”.
- On easy and medium questions, which are within a model’s capabilities, or in settings with small inference requirement, test-time compute can easily cover up for additional pretraining.

Take-home messages

- Test-time and pretraining compute are not 1-to-1 “exchangeable”.
- On easy and medium questions, which are within a model’s capabilities, or in settings with small inference requirement, test-time compute can easily cover up for additional pretraining.
- However, on challenging questions which are outside a given base model’s capabilities or under higher inference requirement, pretraining is likely more effective for improving performance.

Thanks for your listening!

- Q & A