

# Scaling LLM Test-Time Compute Optimally Can be More Effective than Scaling Model Parameters

Charlie Snell

Jaehoon Lee

Kelvin Xu

Aviral Kumar



Google DeepMind

Carnegie Mellon University

# Some Problems Require More Computation at Test Time

## You cannot one-shot everything.

- Challenging problems (e.g. IMO) may require exploring multiple approaches to the problem.
- Even simple problems (e.g. countdown game) may fundamentally require search.

User: Using the numbers [19, 36, 55, 7], create an equation that equals 65.

Assistant: Let me solve this step by step.

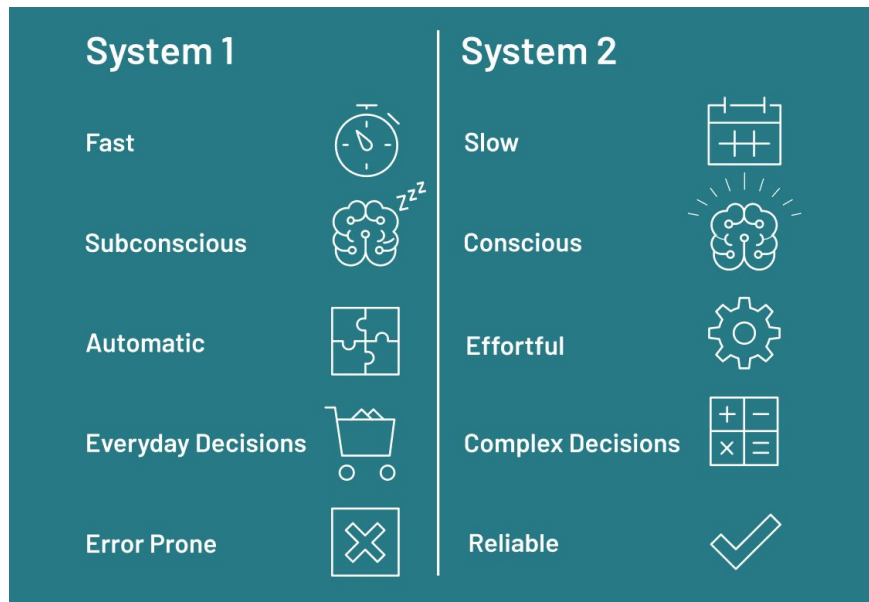
`<think>` We need to find an equation using the numbers 19, 36, 55, and 7 exactly once, with basic arithmetic operations, that equals 65. One possible combination is  $55 + 36 - 19 + 7$ . Let's check:  $55 + 36 = 91$ ,  $91 - 19 = 72$ , and  $72 + 7 = 79$ . However, this doesn't equal 65. Another combination is  $55 + 36 + 7 - 19$ . Let's check:  $55 + 36 = 91$ ,  $91 + 7 = 98$ , and  $98 - 19 = 79$ . This also doesn't equal 65. After trying different combinations, I found that  $55 + 36 + 7 - 19 = 79$ , which is close but not equal to 65. Finally, I found that  $55 + 36 - 19 + 7 = 79$ , which is still not equal to 65. However, if we try  $55 + 36 - 7 + 19$ , we get  $55 + 36 = 91$ ,  $91 - 7 = 84$ , and  $84 + 19 = 103$ , which is not equal to 65. After further trial and error, I found that  $55 + 36 + 7 - 19 = 79$ , which is still not equal to 65. However, if we try  $55 + 36 - 7 - 19$ , we get  $55 + 36 = 91$ ,  $91 - 7 = 84$ , and  $84 - 19 = 65$ . This equals 65. `</think>`

`<answer>`  $55 + 36 - 7 - 19$  `</answer>`

# Analogy to Cognitive Science

## Current LMs are a very strong system 1.

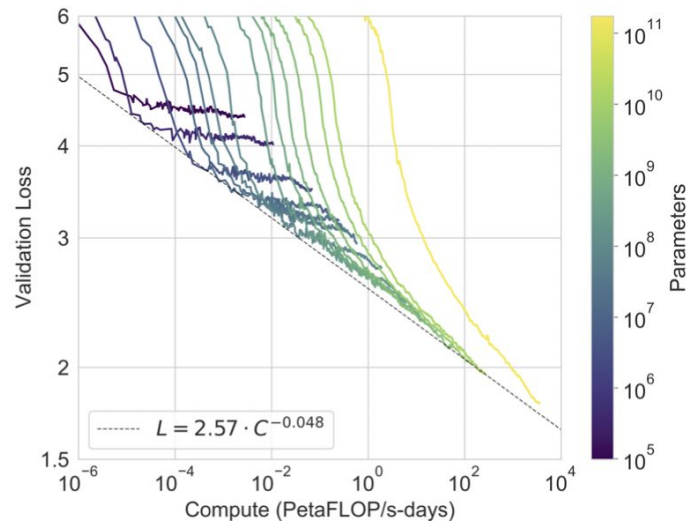
- They can instantly output a coherent essay or a rhyming poem.
- How can we get LMs to think for longer on more challenging problems, like humans do?



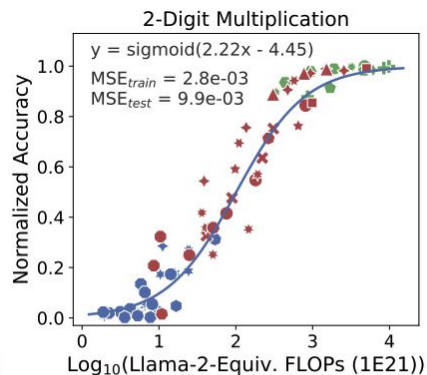
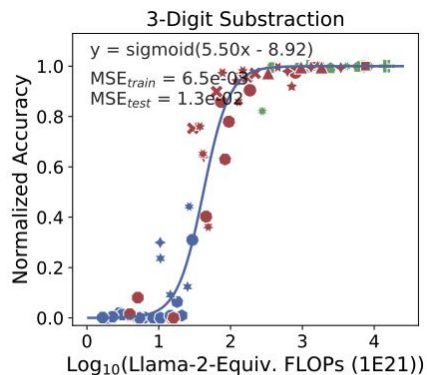
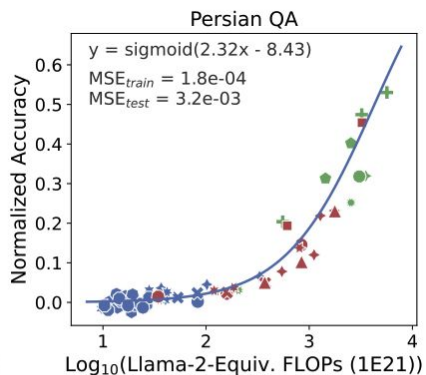
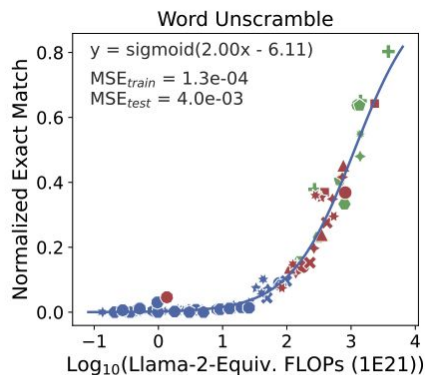
# Classical Scaling Laws

**For a long time, we could only scale parameters.**

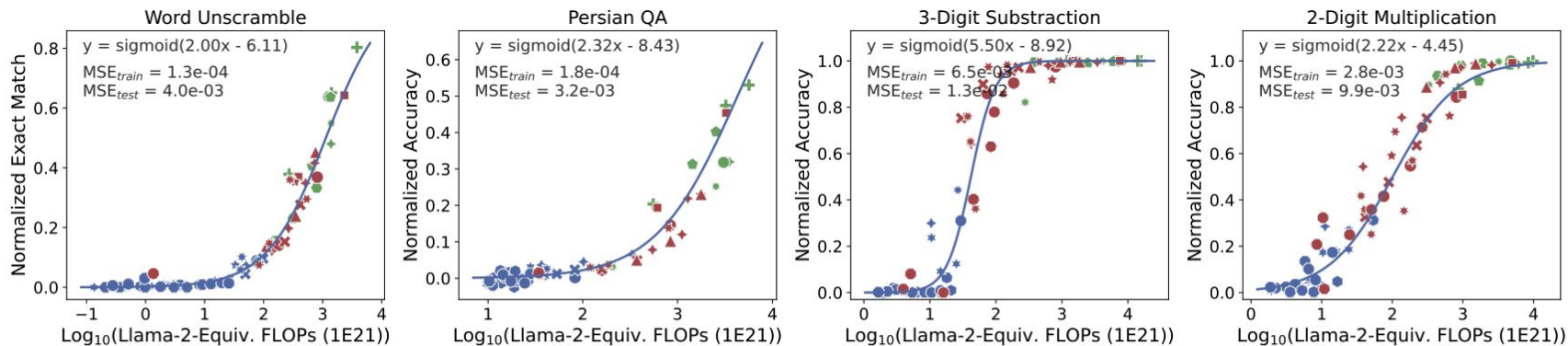
- The more parameters/pretraining compute, the better your next-token prediction ability gets.
- This ability improves predictably with scale.



# Scaling on Downstream Tasks

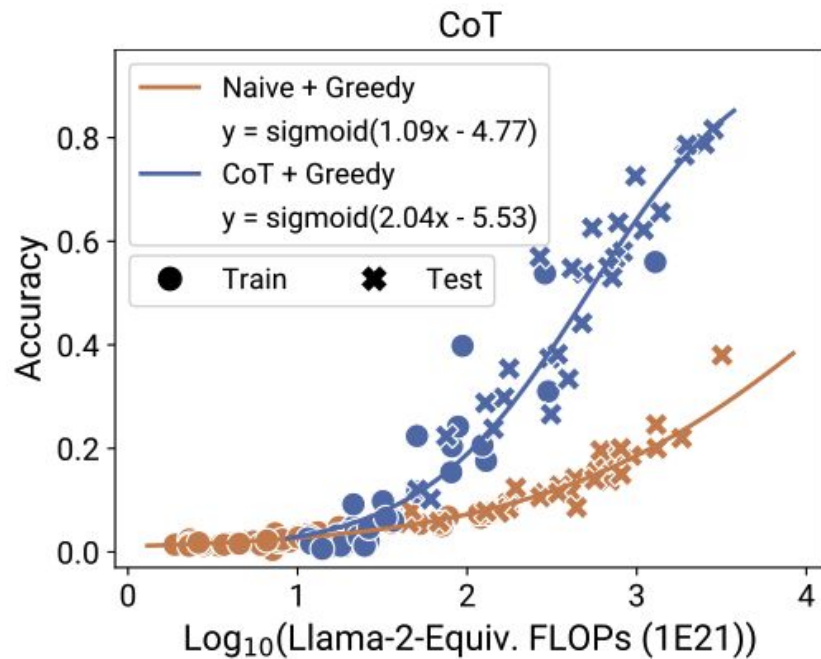


# Scaling on Downstream Tasks

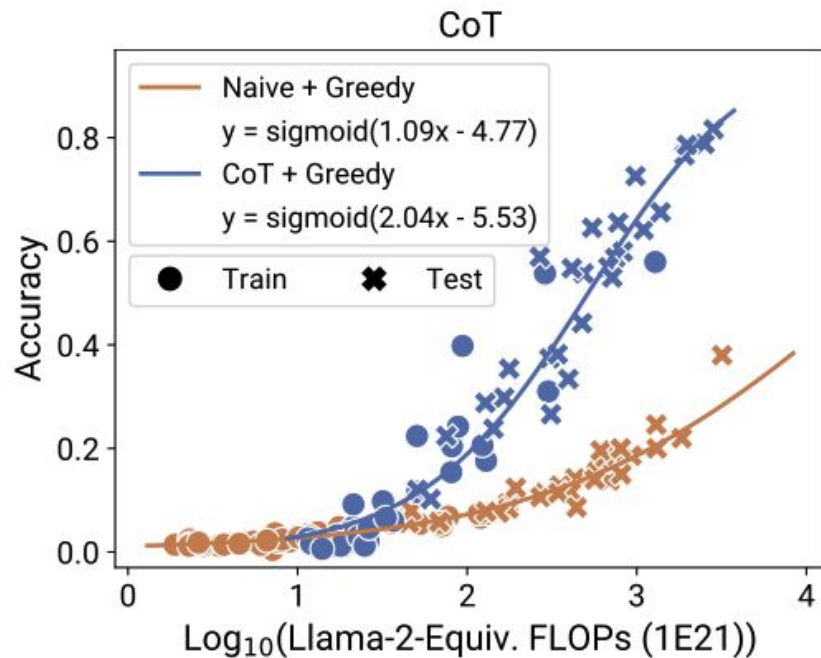


Scaling pre-training yields predictable improvements on many downstream tasks.

# Scaling on Downstream Tasks



# Scaling on Downstream Tasks



**Some downstream tasks scale much more slowly than we would like.**



# Historical Lesson: Test-Time Search Can Improve Scaling

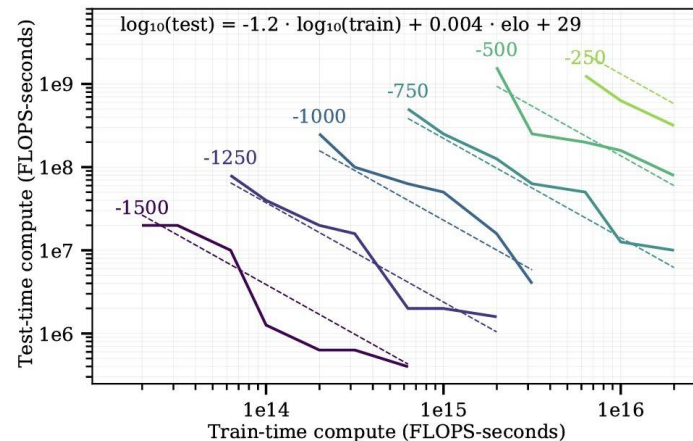
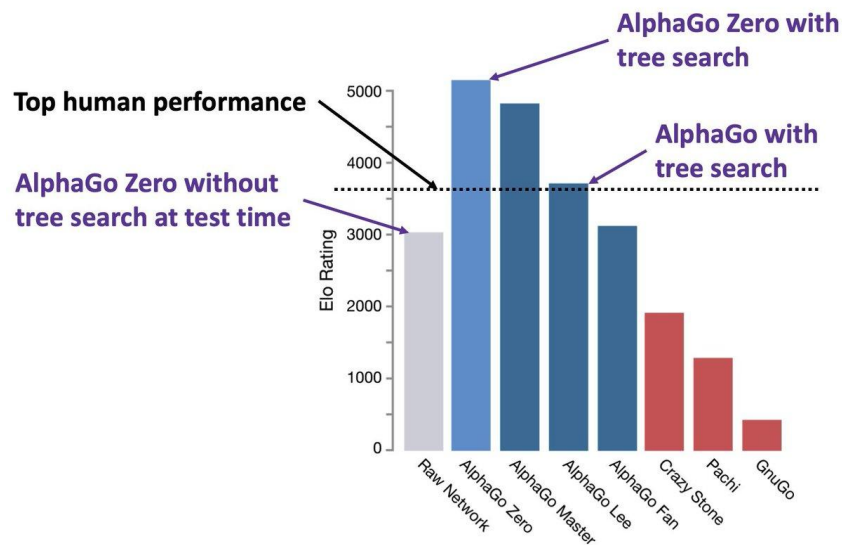
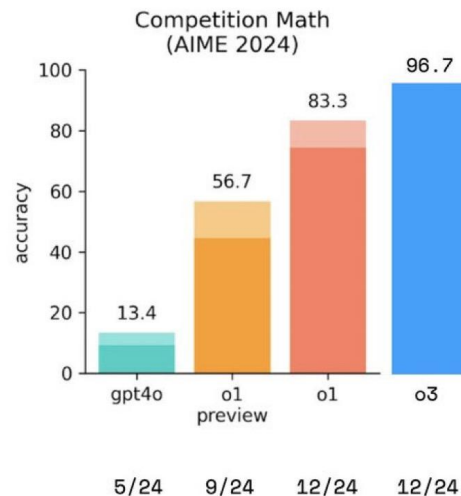
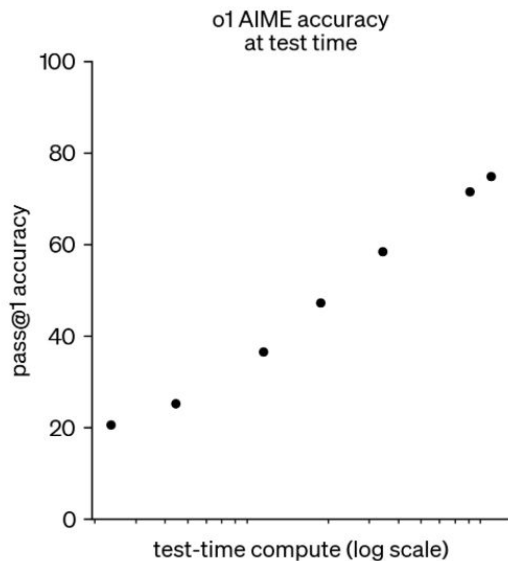
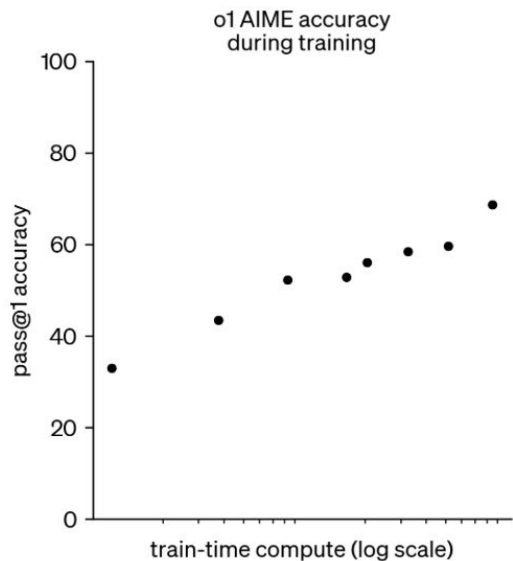


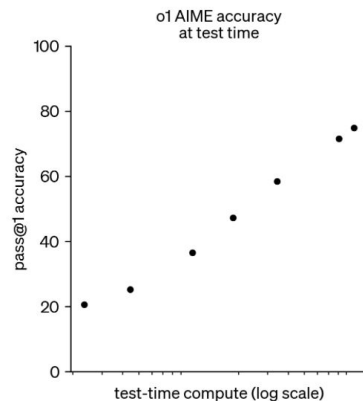
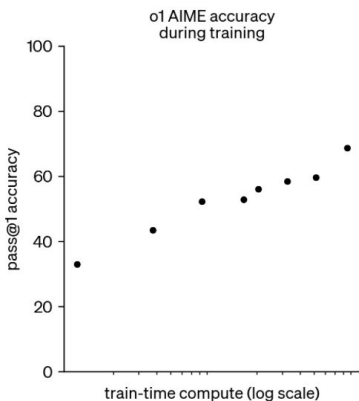
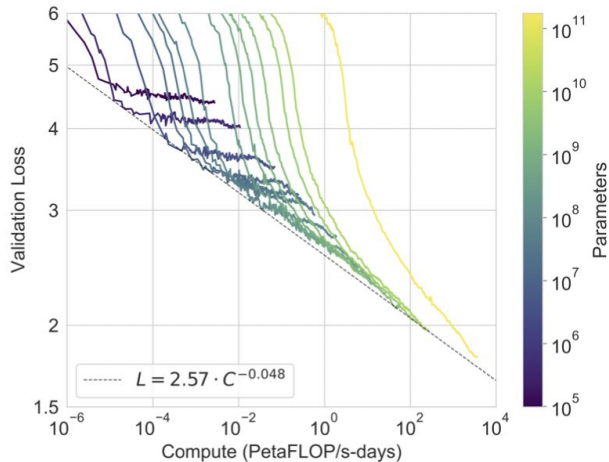
Fig. 9. The trade-off between train-time compute and test-time compute. Each dotted line gives the minimum train-test compute required for a certain Elo on a  $9 \times 9$  board

# Test-Time Scaling with LLMs

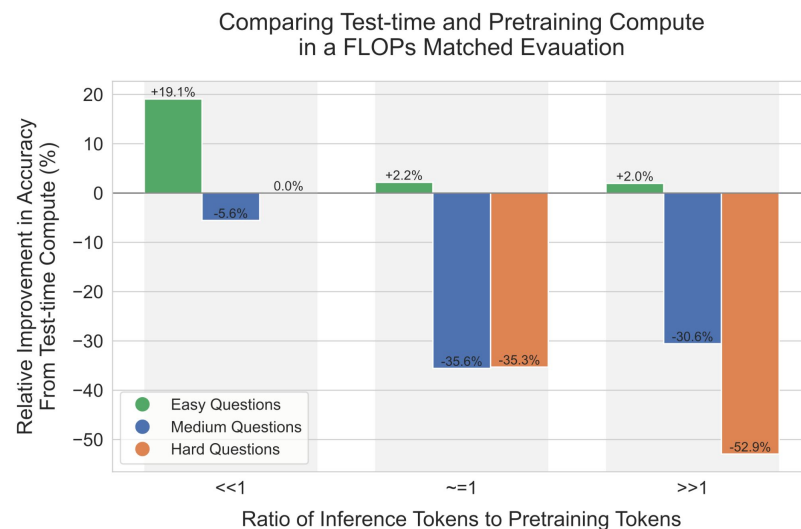
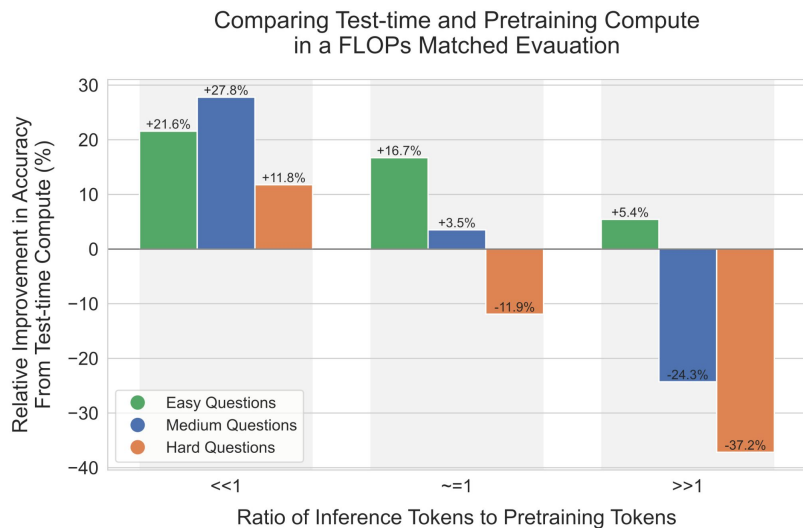


# Pre-Training vs Test-Time Scaling

- Is test-time scaling a good use of compute?
- Would we see larger gains from spending compute doing pre-training instead?



# When can Test-time Compute Outperform Pretraining?



## Our Work: Compute-Optimal Scaling

We select the best test-time compute configuration for a given problem and test-time budget.

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

## Our Work: Compute-Optimal Scaling

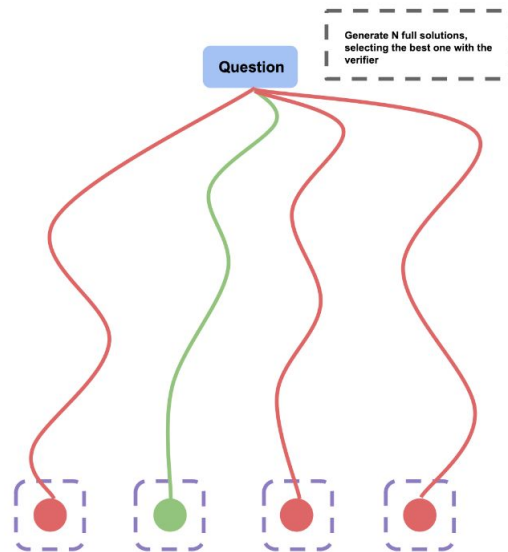
We select the best test-time compute configuration for a given problem and test-time budget.

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

***In practice:*** we select algorithm configurations, such as which search algorithm to use, and use question difficulty as a sufficient statistic to represent the question, instead of specializing the algorithm to each question.

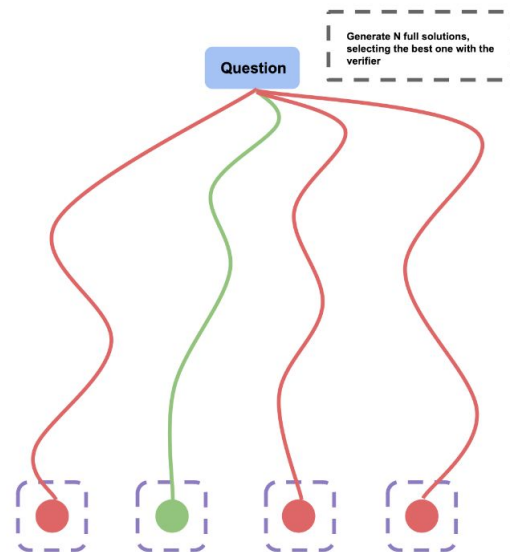
# Compute-Optimal Scaling with Question Difficulty

- We use question difficulty as a sufficient statistic for practically estimating compute optimal scaling.
  - For a given difficulty level, select the best performing hyperparameters.



# Compute-Optimal Scaling with Question Difficulty

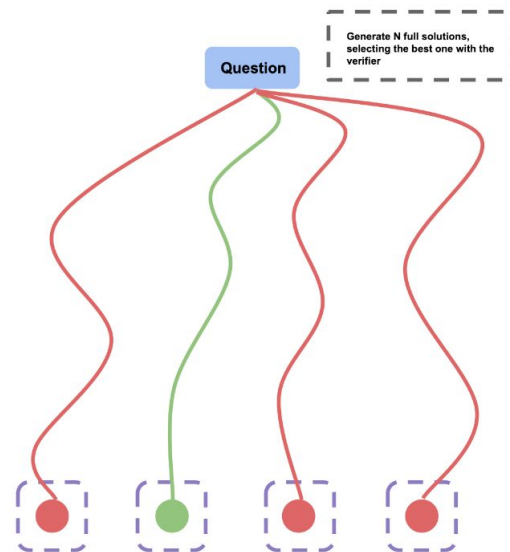
- We use question difficulty as a sufficient statistic for practically estimating compute optimal scaling.
  - For a given difficulty level, select the best performing hyperparameters.
- **Oracle difficulty**
  - Sample 2048 outputs per question and bin questions into 5 quantiles by how many of the samples were correct.





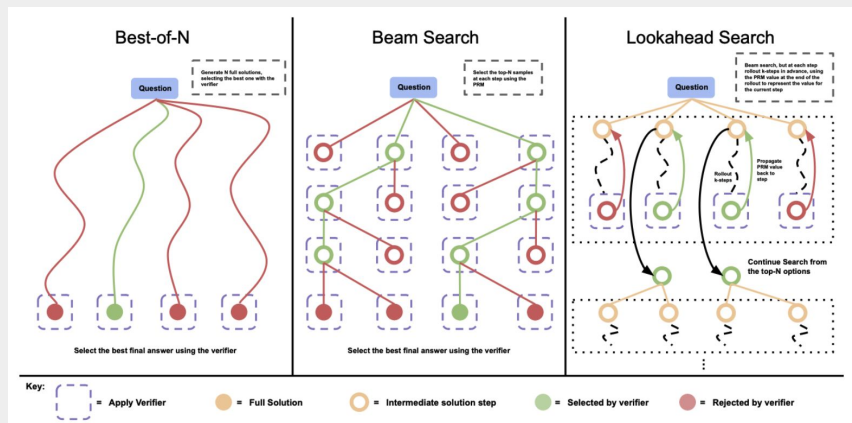
# Compute-Optimal Scaling with Question Difficulty

- We 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.

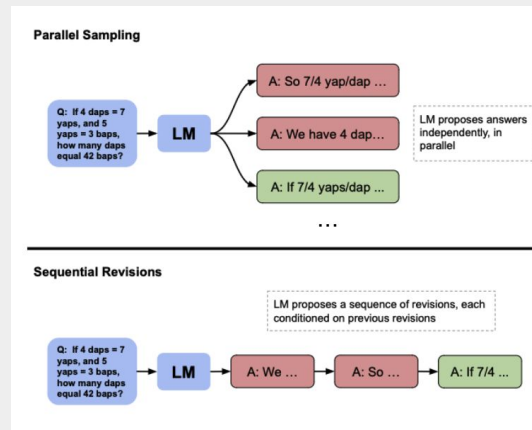


# Two Basic Approaches to Scaling Test-Time Compute

## Searching Against a Verifier

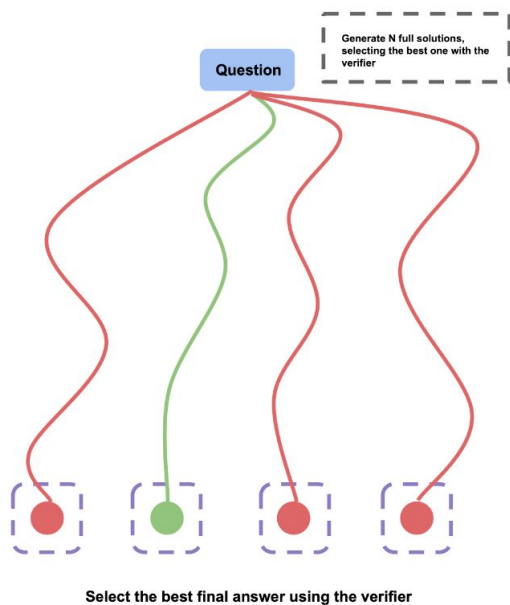


## Revising Answers Iteratively

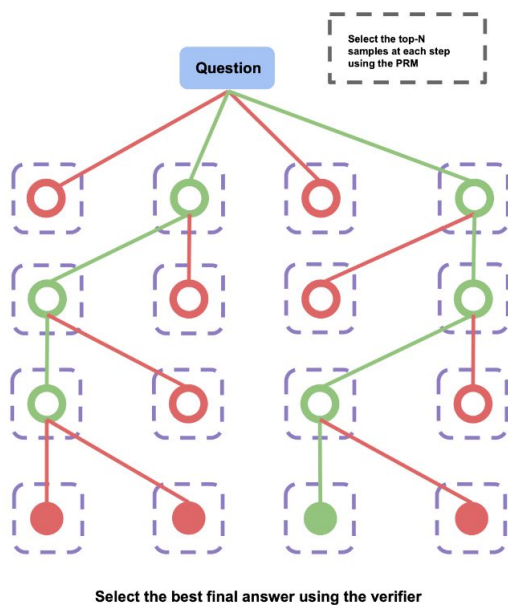


# Scaling Test-time Compute With Verifiers

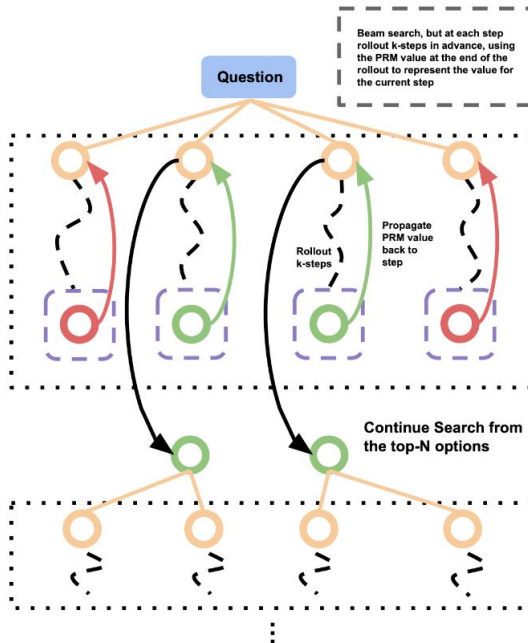
## Best-of-N



## Beam Search



## Lookahead Search



Key:



= Apply Verifier



= Full Solution



= Intermediate solution step

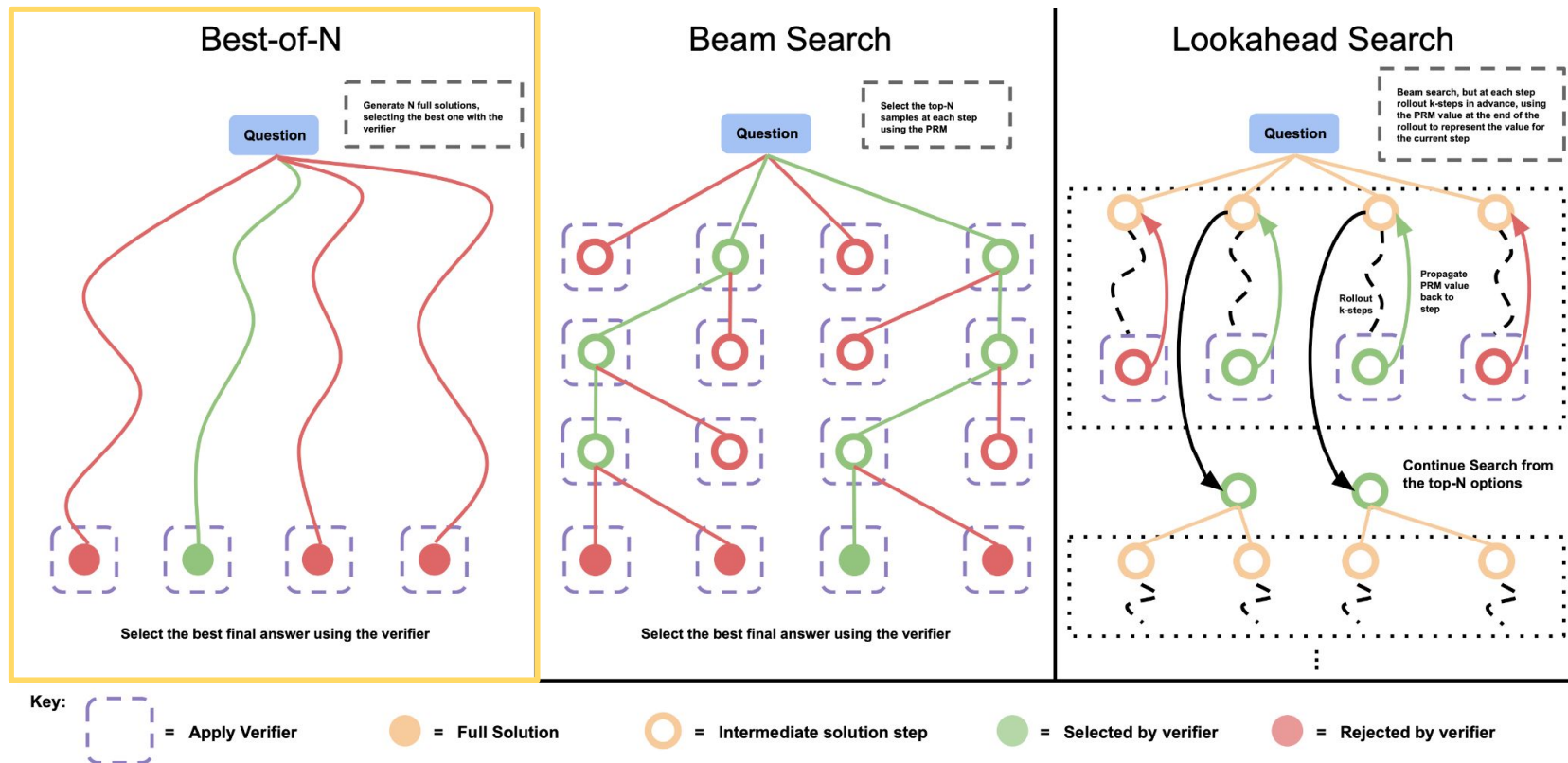


= Selected by verifier



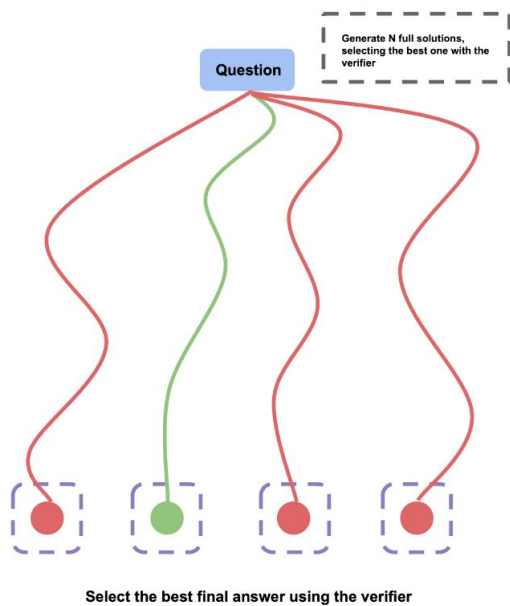
= Rejected by verifier

# Scaling Test-time Compute With Verifiers

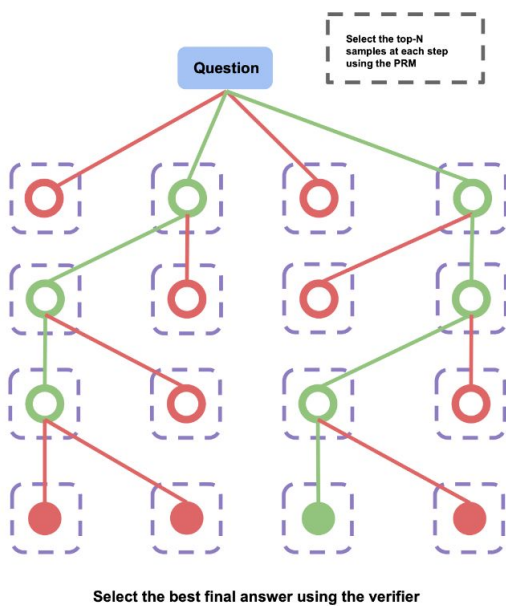


# Scaling Test-time Compute With Verifiers

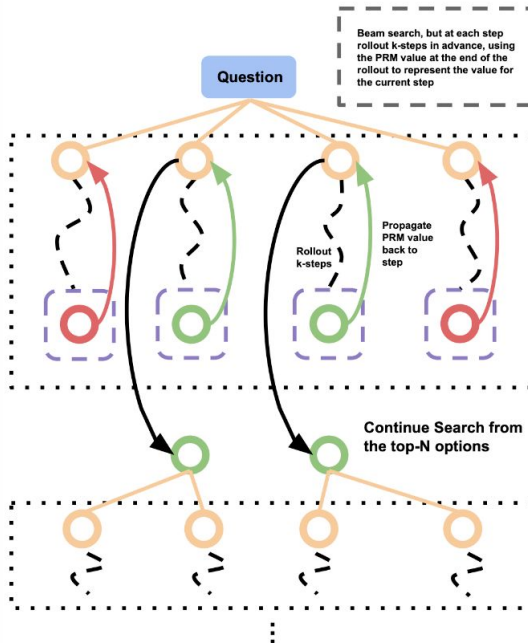
## Best-of-N



## Beam Search



## Lookahead Search



Key:



= Apply Verifier



= Full Solution



= Intermediate solution step

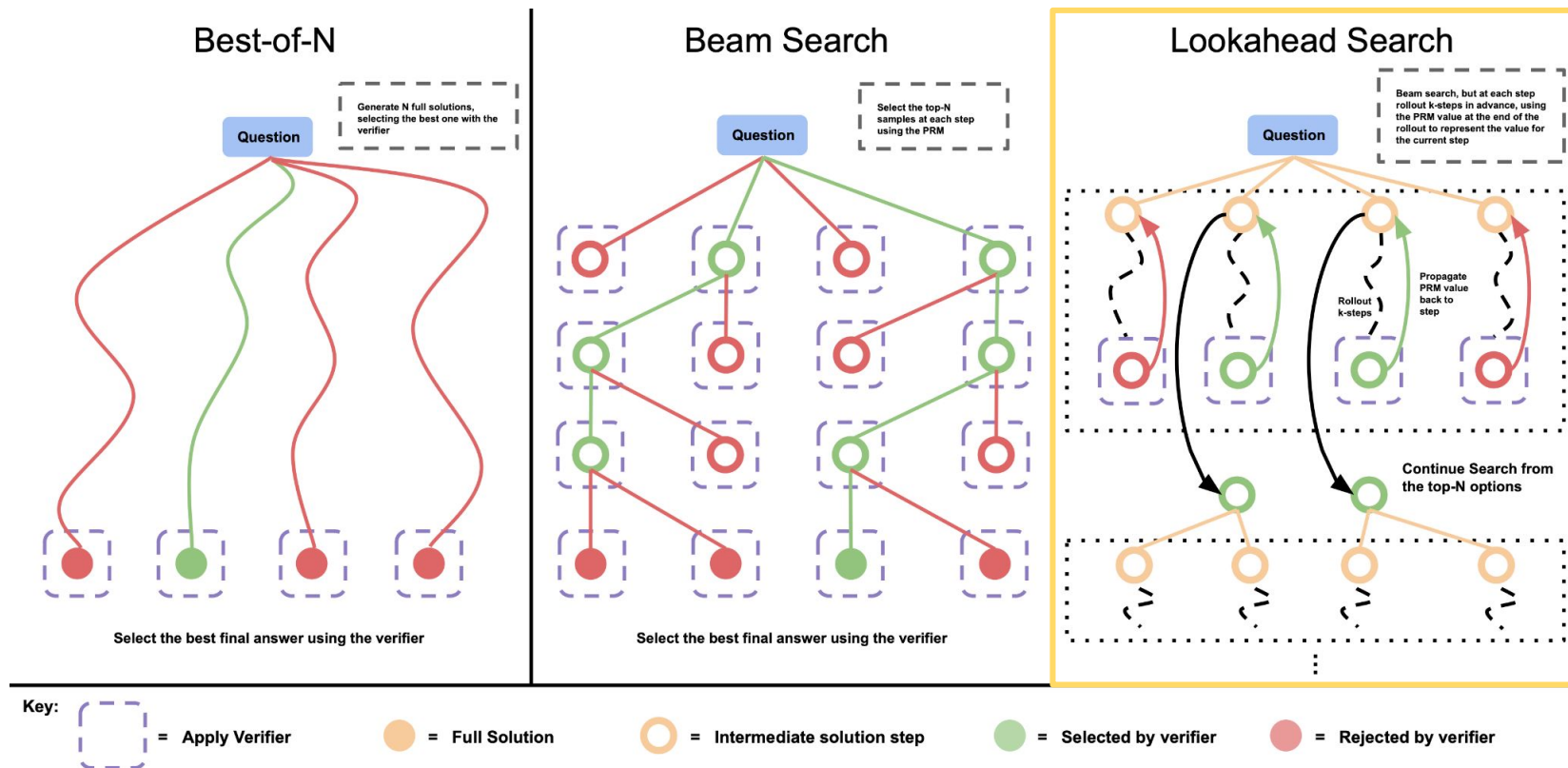


= Selected by verifier



= Rejected by verifier

# Scaling Test-time Compute With Verifiers

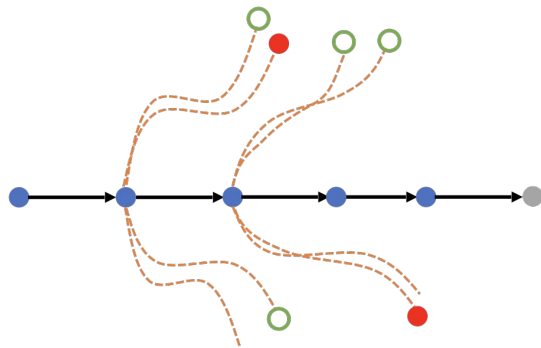


# Training a Process Based Verifier

- We learn a PRM, which is a verifier that scores each step in a solution.
  - Prior work [\[lightman et al. 2023\]](#) used human labels of per-step correctness for supervision.
  - We instead follow the approach from MathShepard [\[wang et al. 2024\]](#) to learn a value function.

# Training a Process Based Verifier

- We learn a PRM, which is a verifier that scores each step in a solution.
  - Prior work [\[lightman et al. 2023\]](#) used human labels of per-step correctness for supervision.
  - We instead follow the approach from MathShepard [\[wang et al. 2024\]](#) to learn a value function.

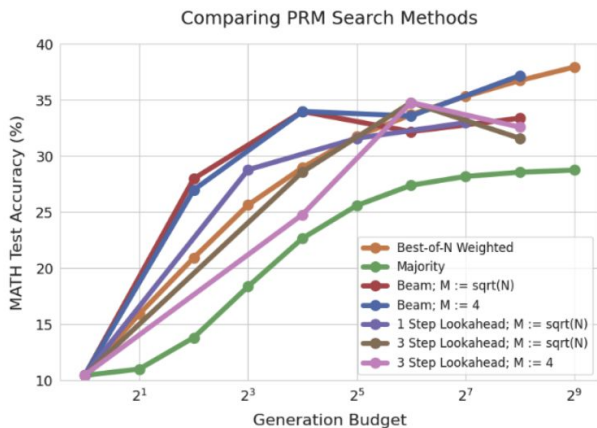


- Rollout N times from each step in the solution, get the correctness rate of the rollouts
- Train the PRM on these MC rollout targets



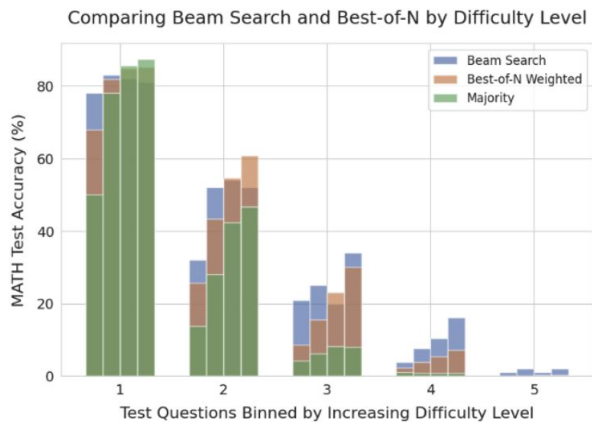
# Comparing Search Methods

- 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.

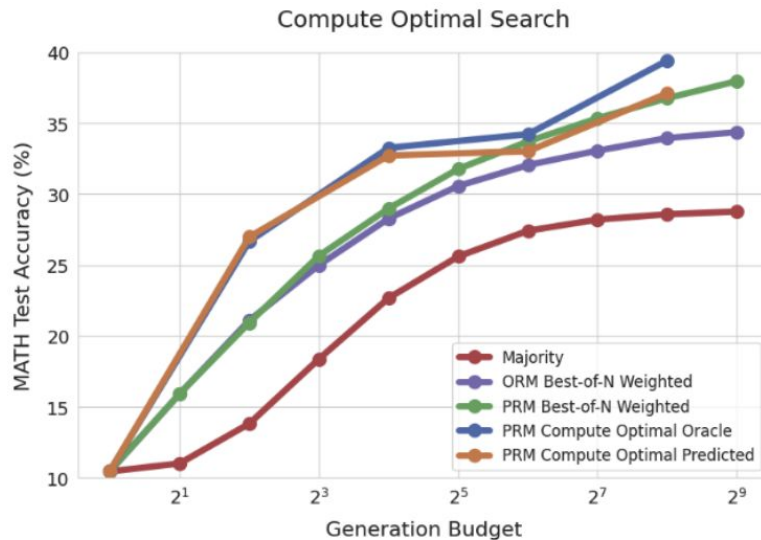


# Performance Breakdown by Question Difficulty

- 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



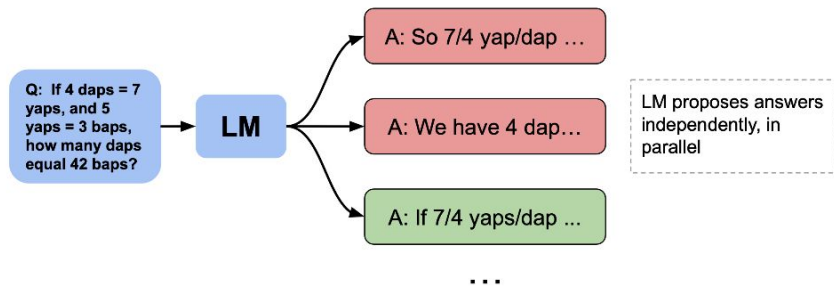
By selecting the best performing search algorithm at each difficulty level, we can nearly outperform best-of-N using up to **4x** less test-time compute.

# Takeaways for Scaling Test-time Compute With Verifiers

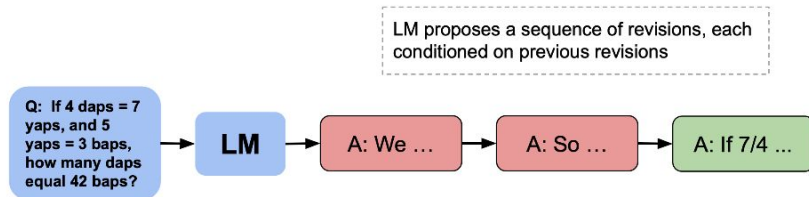
- The efficacy of a search method depends on the budget and the question.
- Beam search is more effective on harder questions and at lower budgets.
- Best-of-N is more effective on easier questions and at higher budgets.
- By selecting the best setting for each question, we can nearly outperform best-of-N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

# Scaling Test-time Compute With Revisions

## Parallel Sampling



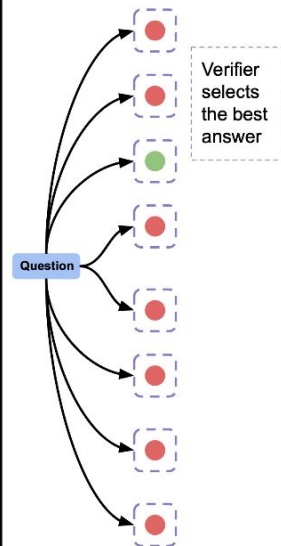
## Sequential Revisions



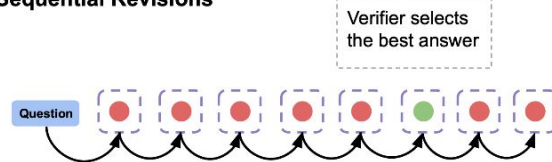
## Using Revision Model + Verifier at Inference Time



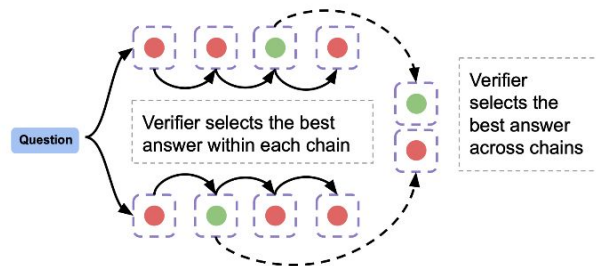
### Parallel Best-of-N



### Sequential Revisions



### Combining Sequential / Parallel

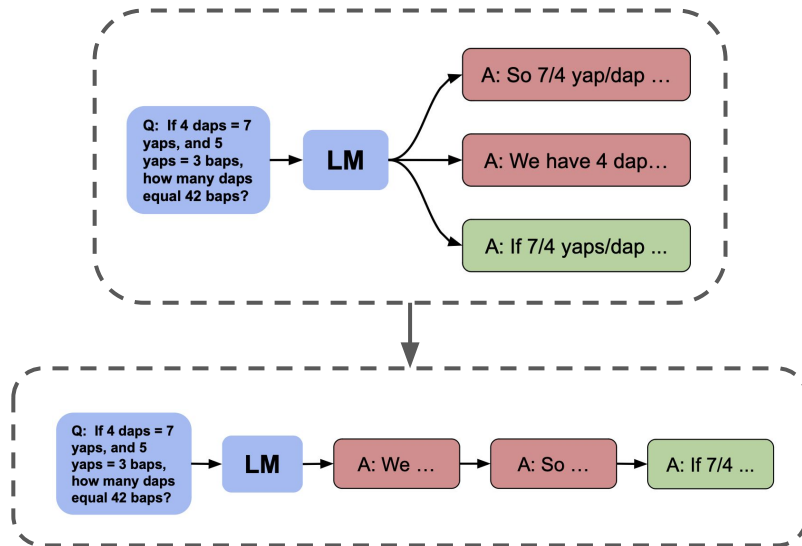


# Finetuning a Revision Model

We finetune 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.

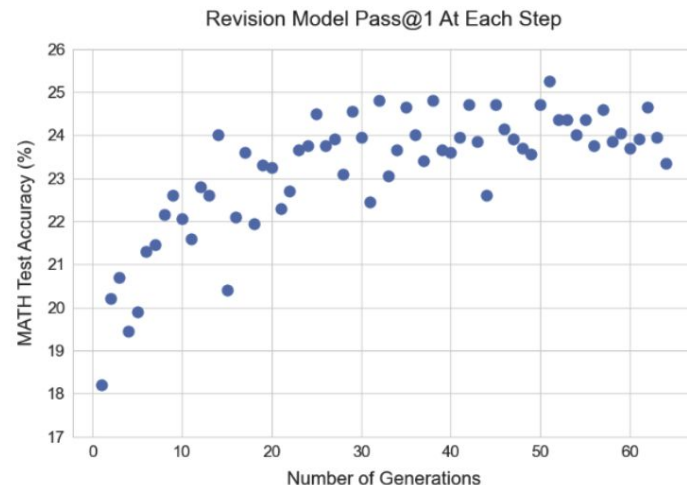


# Finetuning a Revision Model

We finetune 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.



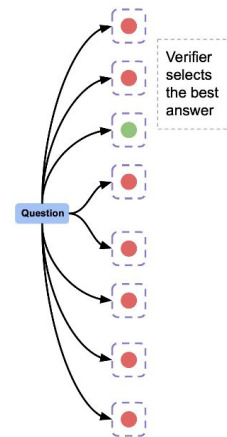
# Using a Verifier with the Revision Model

- Sometimes the model will “revise” a correct answer into an incorrect one.
- **To correct for this, we:**
  - sample N outputs in sequence
  - use a verifier to select the most likely correct answer from the chain

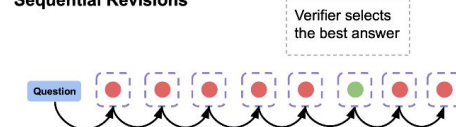
Using Revision Model + Verifier at Inference Time



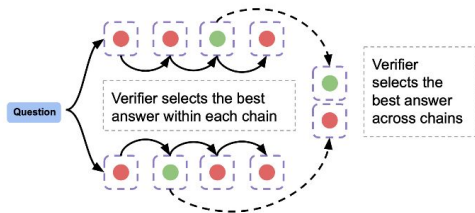
Parallel Best-of-N



Sequential Revisions

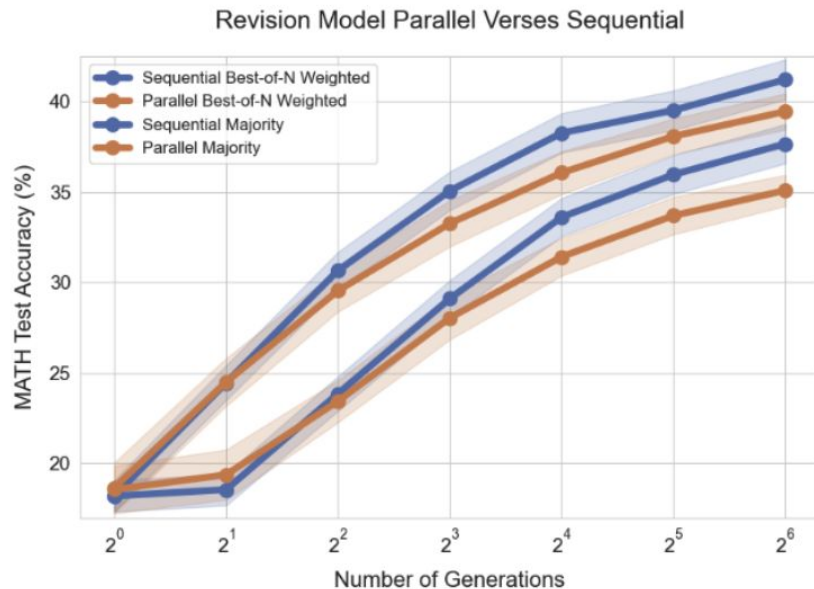


Combining Sequential / Parallel



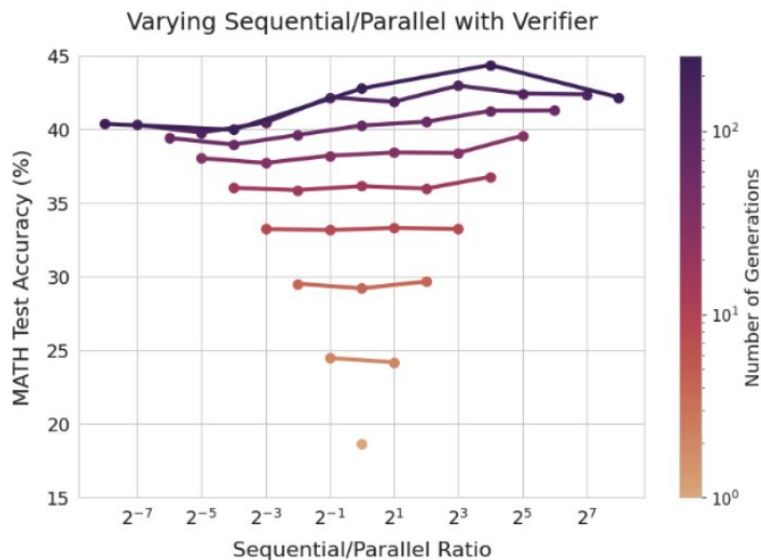


# Comparing Sequential and Parallel Sampling



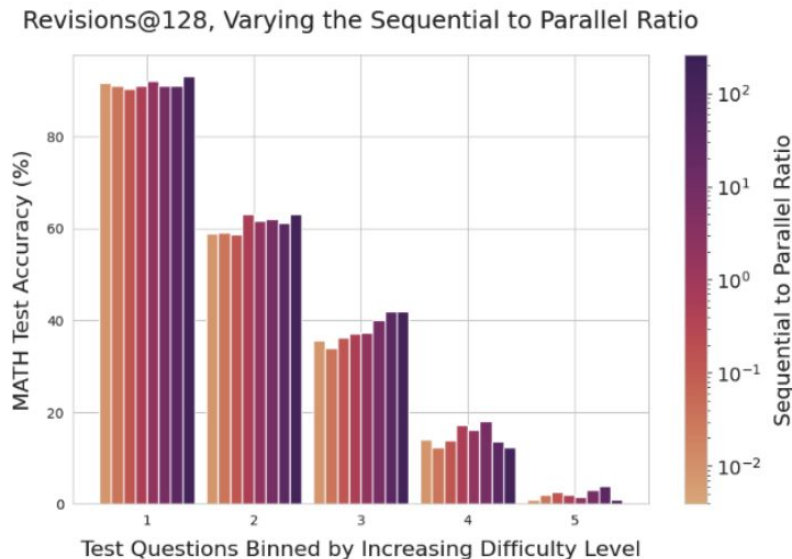
Sampling N outputs in sequence from the model outperforms sampling N in parallel.

# Comparing Sequential and Parallel Sampling



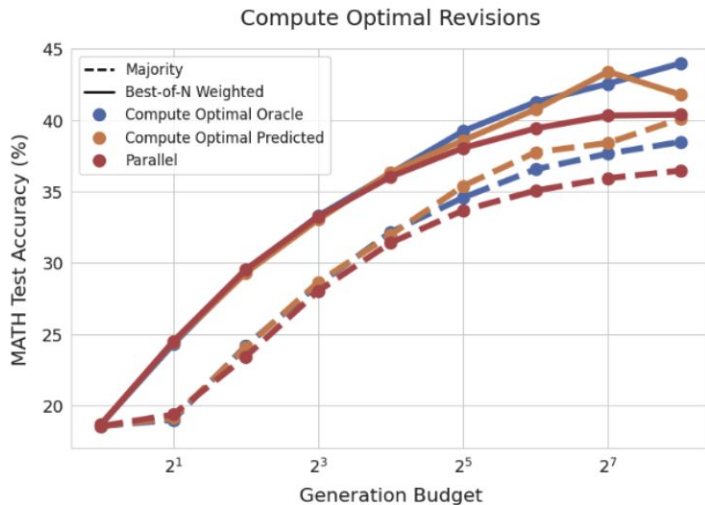
In some cases there is an ideal ratio of sequential to parallel test-time compute.

# Comparing Sequential and Parallel Sampling



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

# Scaling Test-time Compute With Revisions

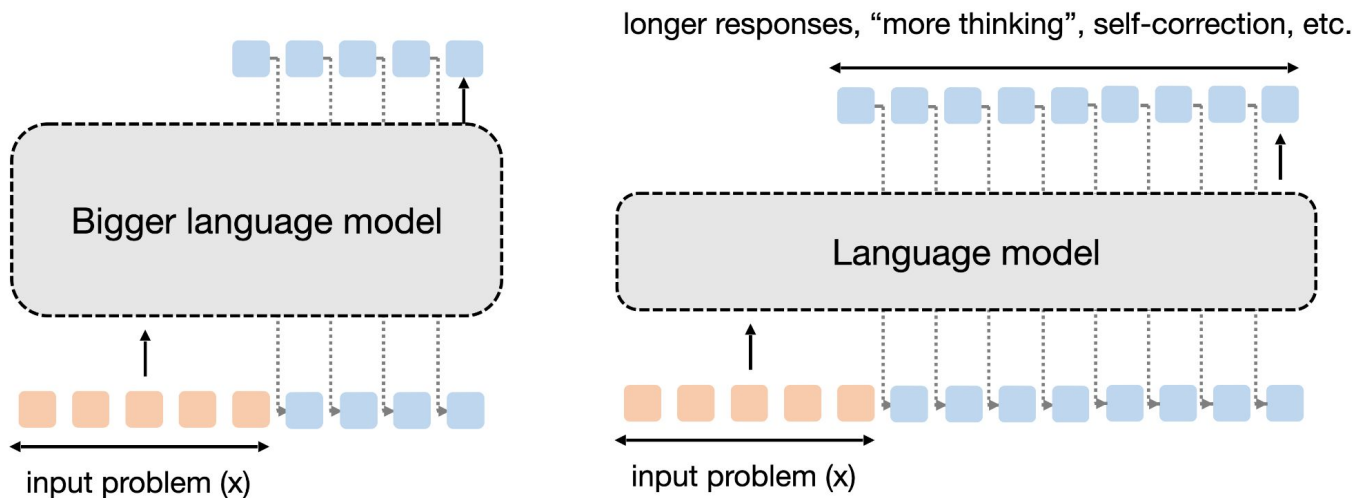


By selecting the best performing ratio at each difficulty level, we can outperform parallel sampling using up to **4x** less test-time compute.

# Takeaways for Scaling Test-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.

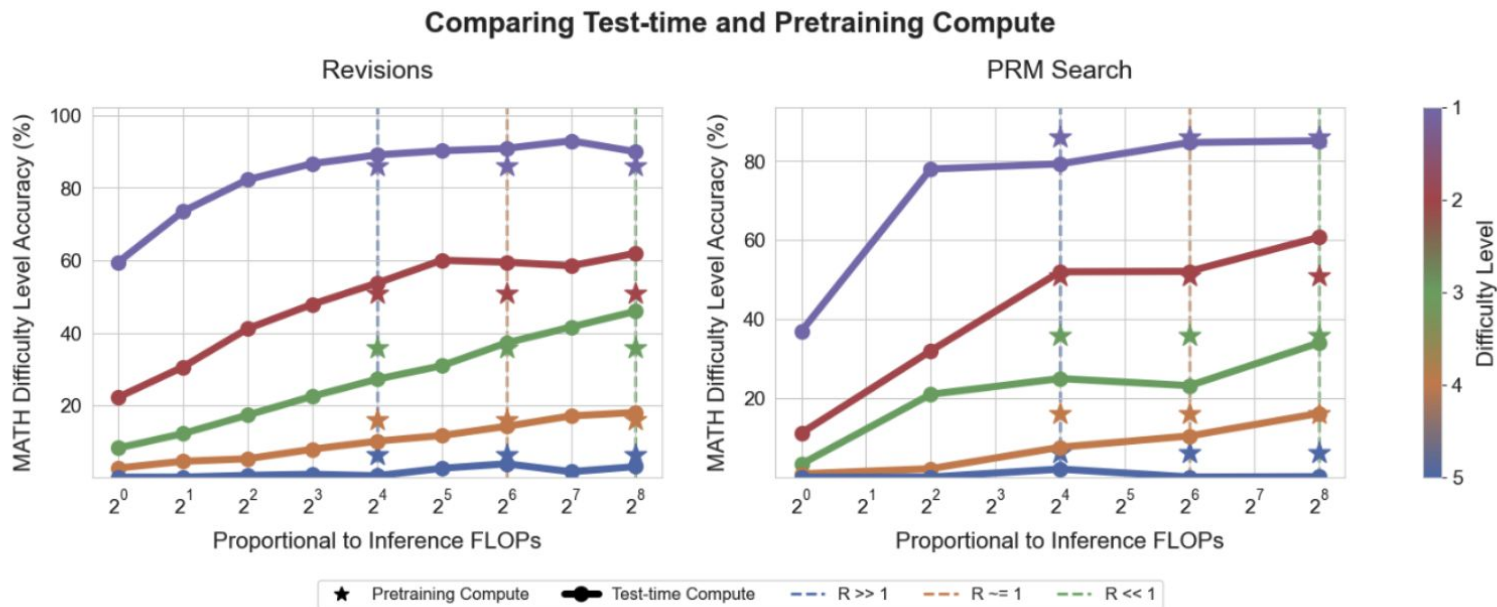
# Exchanging Test-time and Pretraining Compute



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

- Depends on the number of inference queries we are expecting.
- If serving to many users, then extra test-time compute will cost a lot.
- If running few inferences, then we can afford more test-time compute.

# Exchanging Test-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.**

# Takeaways/Discussion

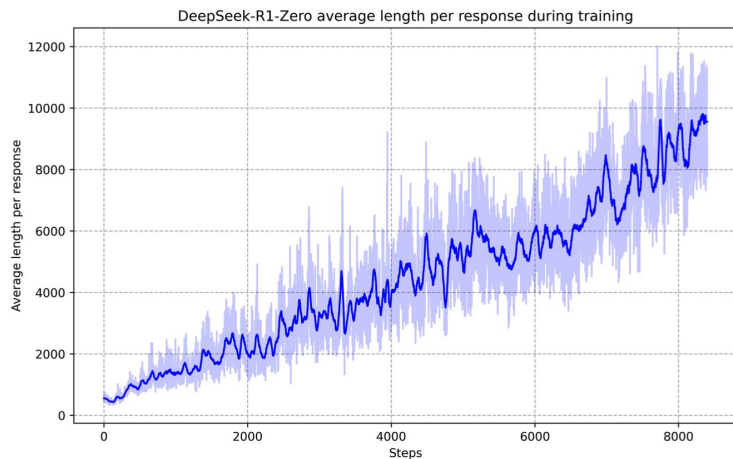
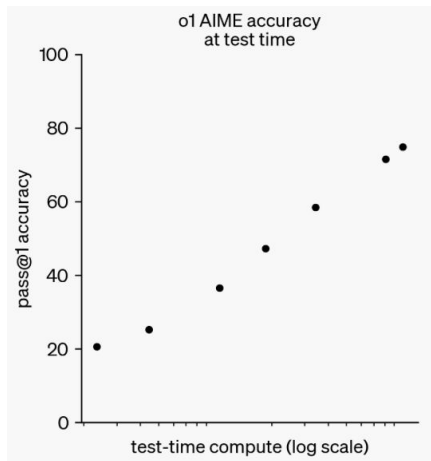
**Using fairly simple methodology we find that scaling LLM test-time compute can greatly improve performance, and in some cases it can outperform scaling parameters.**



# Takeaways/Discussion

There is much room for future work to:

1. Improve upon our techniques and explore alternative approaches to scaling test-time compute.



# Takeaways/Discussion

There is much room for future work to:

1. Improve upon our techniques and explore alternative approaches to scaling test-time compute.
2. Conduct additional analysis. E.g. how does the scale of finetuning impact test-time scaling?

