SCALING LLM TEST-TIME COMPUTE OPTIMALLY CAN BE MORE EFFECTIVE THAN SCALING PARAMETERS FOR REASONING

Charlie Snell[*], **Jaehoon Lee**[§], **Kelvin Xu**[§†], **Aviral Kumar**[#§†]

ABSTRACT

Enabling LLMs to improve their outputs by using more test-time compute is a critical step towards building self-improving agents that can operate on open-ended natural language. In this paper, we scale up inference-time computation in LLMs, with a focus on answering: *if an LLM is allowed to use a fixed but non-trivial amount of inference-time compute, how much can it improve its performance on a challenging prompt?* Answering this question has implications not only on performance, but also on the future of LLM pretraining and how to tradeoff inference-time and pre-training compute. Little research has attempted to understand the scaling behaviors of test-time inference methods, with current work largely providing negative results for a number of these strategies. In this work, we analyze two primary mechanisms to scale test-time computation: **(1)** searching against dense, process-based verifier reward models (PRMs); and **(2)** updating the model's distribution over a response adaptively, given the prompt at test time. We find that in both cases, the effectiveness of different approaches to scaling test-time compute critically varies depending on the difficulty of the prompt. This observation motivates applying a "**compute-optimal**" scaling strategy, which acts to, as effectively as possible, allocate test-time compute per prompt in an adaptive manner. Using this compute-optimal strategy, we can improve the efficiency of test-time compute scaling for math reasoning problems by more than **4**× compared to a best-of-N baseline. Additionally, in a FLOPs-matched evaluation, we find that on problems where a smaller base model attains somewhat non-trivial success rates, test-time compute can be used to outperform a **14**× larger model.

Presenters:

Anant Gupta,
Jane Yijun Sun

What is Test-Time Compute?

- The compute used by LLM to “think” during inference.
  - Chain of Thought (CoT) is a type of Test-Time Compute.
- Verifier based methods use another LLM to judge the outputs.
  - Process Reward Models (PRM)
  - Outcome Reward Models (ORM)
- Different output generation techniques guided by RM -
  - Beam Search
  - Best-of-N
  - Lookahead-Search

Classifying Test-Time Compute (TTC)

- TTC can be interpreted as modifying the LLM distribution.
- This leads to 2 disjoint categories -
  - **Optimizing the verifier** (PRM or ORM)
  - **Modifying the proposal distribution**
    - E.g. prompting models to refine their own reasoning by self reflection.

What combination of these works the best?

Fundamental Questions

Given a prompt and a test-time compute budget.

- How can we determine the most effective way to utilize test-time compute for a given prompt?
- How well would this do against simply utilizing a much bigger pretrained model?

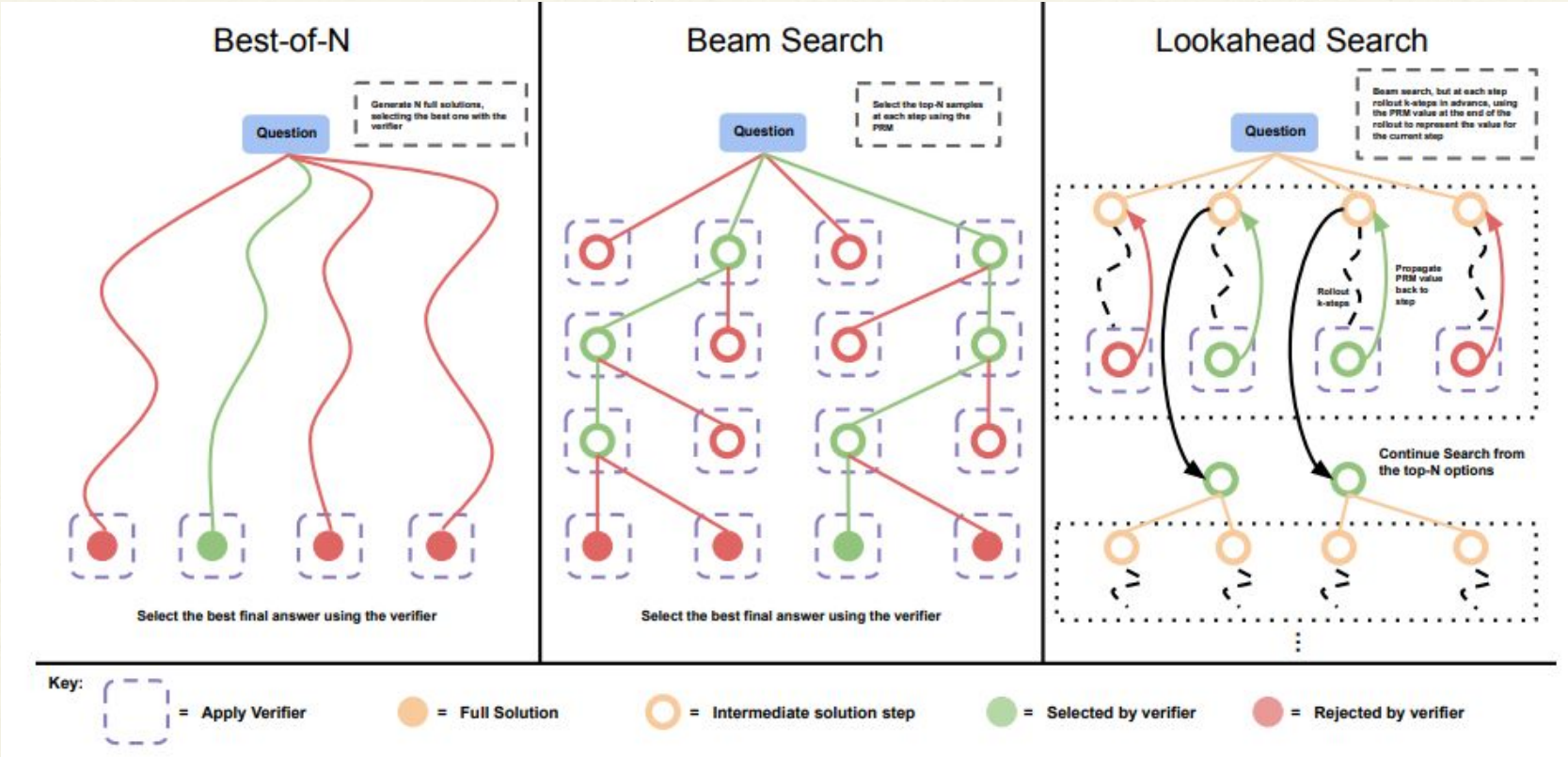
The Hypothesis

- **Hypothesis -** Question difficulty is all you need to decide optimal test-time compute strategy.
- **Definitions -**
  - Question Difficulty
    - The proportion of correct answer out of 2048 samples.
    - Create 5 different quantile bins
  - Oracle Difficulty
    - Correct answers are provided by an oracle
  - Model Predicted
    - Correct answers are the average score of verifiers.

Experimental Setup

- **Dataset - HendrycksMath (Use only one dataset)**
- **Model - PaLM 2-S* (Unknown size but estimated 10B-100B)**
- Constant compute across all directly compared methods

Which Verifier Based Method is Best?

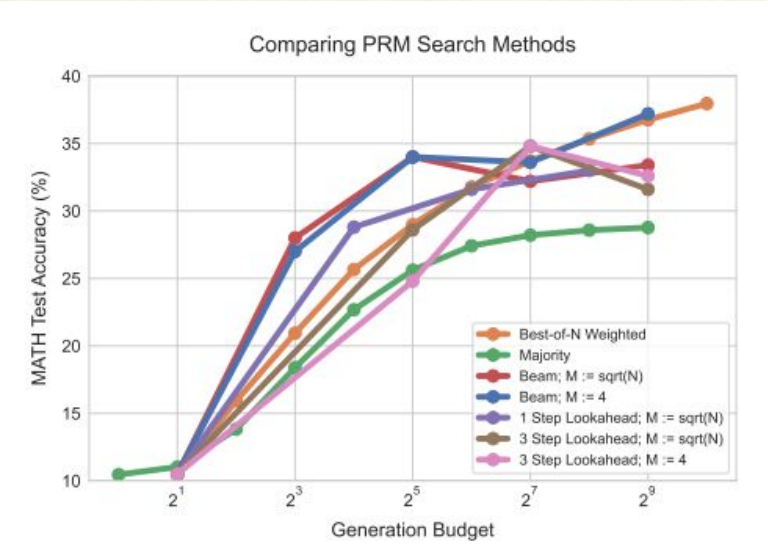


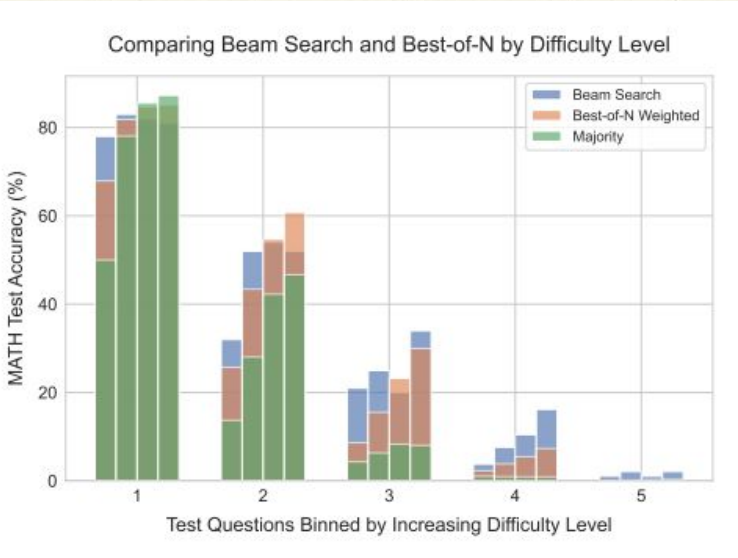
Which Verifier Based Method is Best?

- **Best-of-N weighted.**
  - Sample N answers independently from the base LLM and then select the best answer according to the PRM's final answer judgment.
- **Beam search.**
  - Beam search optimizes the PRM by searching over its per-step predictions.
  - Consider a fixed number of beams N and a beam width M.
  - Apply best-of-N weighted selection to make our final answer prediction.
- **Lookahead search.**
  - At each step in Beam search, rather than using the PRM score at the current step to select the top options, perform a simulation, rolling out k steps.

Verifier Based Method – All Experiments

- **Evaluate methods under a fixed compute budget (≤ 256)**
- **Beam search settings:**
  - Beam width = sqrt(N)
  - Fixed beam width M = 4
- **Lookahead search adds extra compute:**
  - Cost ≈ N × (k + 1)
- **Verifier usage introduces ~2× overhead across methods**
- **Results:**
  - Beam search performs better at low budgets
  - Best-of-N performs better at high budgets
- **Lookahead search underperforms due to higher compute cost**

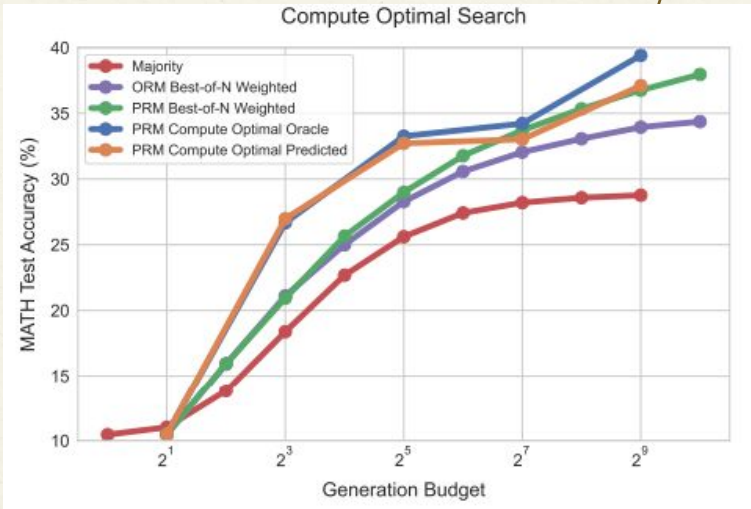




Top Left - Ablations as described previously. For Beam, M=4 is chosen.

Top Right - Comparing different methods for 4,16,64, 256 generations on each difficulty level. Beam Search is over-optimized on higher budgets.

Bottom - Shows that PRM optimal uses 4x less compute for same accuracy (16 vs 64)



Verifier Based Method – Results

- **Performance varies strongly with question difficulty**
- **On easy questions (levels 1–2):**
  - Beam search hurts performance at higher budgets
  - Likely due to over-optimization of PRM
- **On medium/hard (levels 3–4):**
  - Beam search outperforms Best-of-N
- **On very hard (level 5):**
  - No method improves meaningfully
- **Key issue: search can exploit verifier errors**
  - e.g., repetitive or overly short solutions

Verifier Based Method – Takeaways

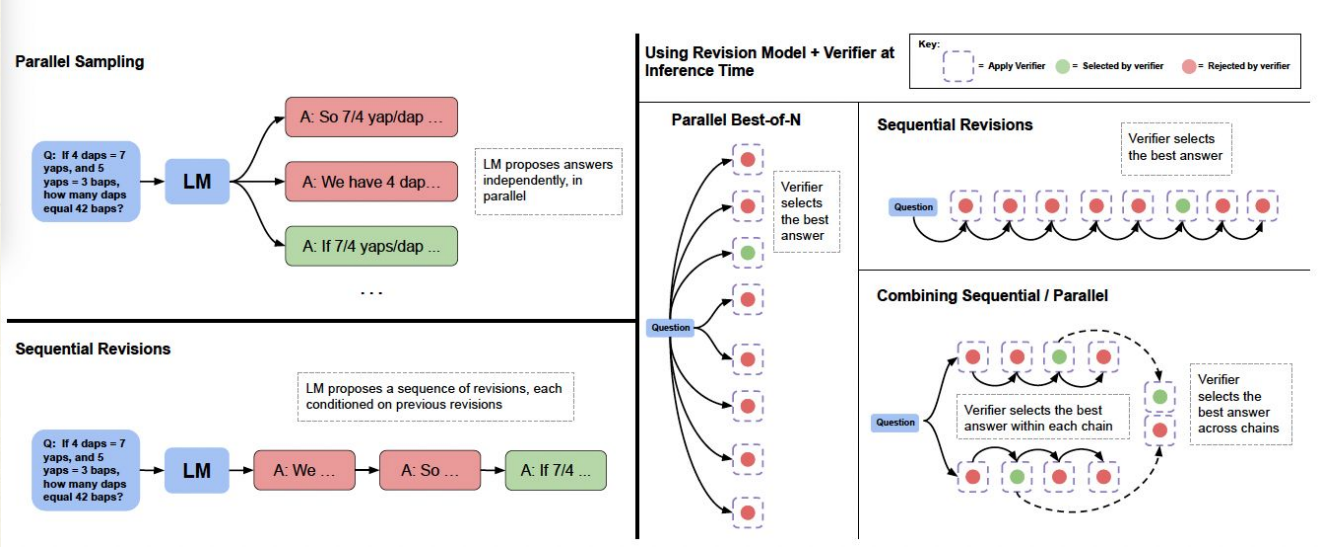
- **Optimal method depends on:**
  - Difficulty + compute budget
- **Strategy:**
  - Use beam search for harder problems / low budgets
  - Use Best-of-N for easier problems / high budgets
- **Adaptive allocation → "compute-optimal scaling"**
- **Using difficulty bins:**
  - Achieves similar or better performance with up to 4× less compute
- **Works with:**
  - Oracle difficulty (ground truth) and
  - Predicted difficulty (PRM-based)
- **Key takeaway:**
  - Allocate test-time compute adaptively per problem

Beyond Best-of-N: Iterative Revisions

Goal:
Improve the model's distribution by allowing it to "think again" and fix its own mistakes.

Challenge:
Prompting off-the-shelf LLMs to self-correct is often ineffective for reasoning.

Solution:
Specifically fine-tune models to iteratively revise answers using trajectories of incorrect-to-correct transitions.



Training the Revision Model

Data

Constructed from 64 parallel samples to find sequences of up to 4 incorrect attempts followed by a correct revision.

Supervised Fine-Tuning (SFT)

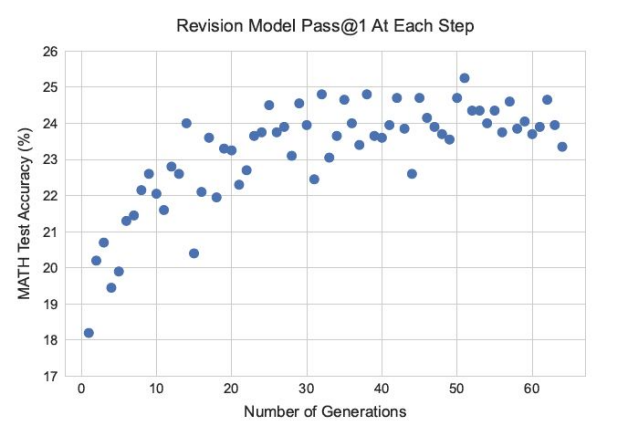
Teaching the model to identify and edit errors rather than starting from scratch.

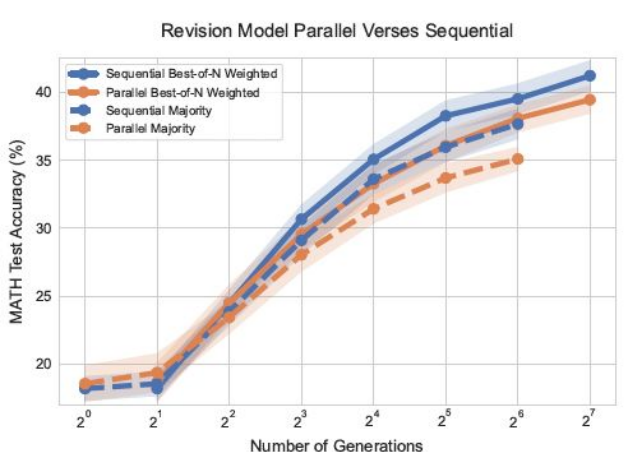
Inference Shift

At test-time, the model can sample longer chains than the 4-step limit it was trained on.

Evaluating Revision Performance

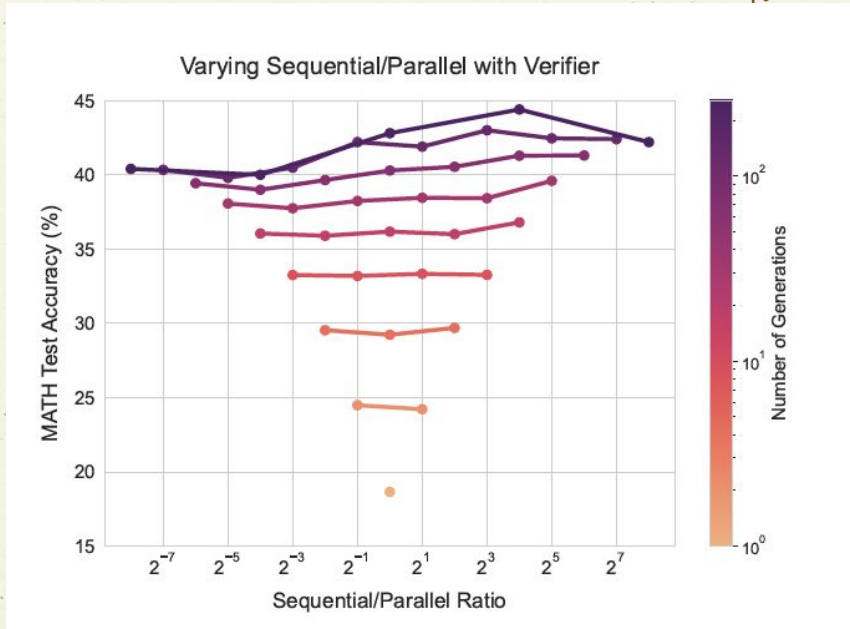
- **Pass@k Improvement:**
  - Accuracy increases as the revision chain length grows.
- **The Conversion Risk:**
  - Approximately **38%** of correct answers are accidentally changed to incorrect during revision.
- **The Solution:**
  - Use a verifier or majority voting to select the best answer from the entire sequence.





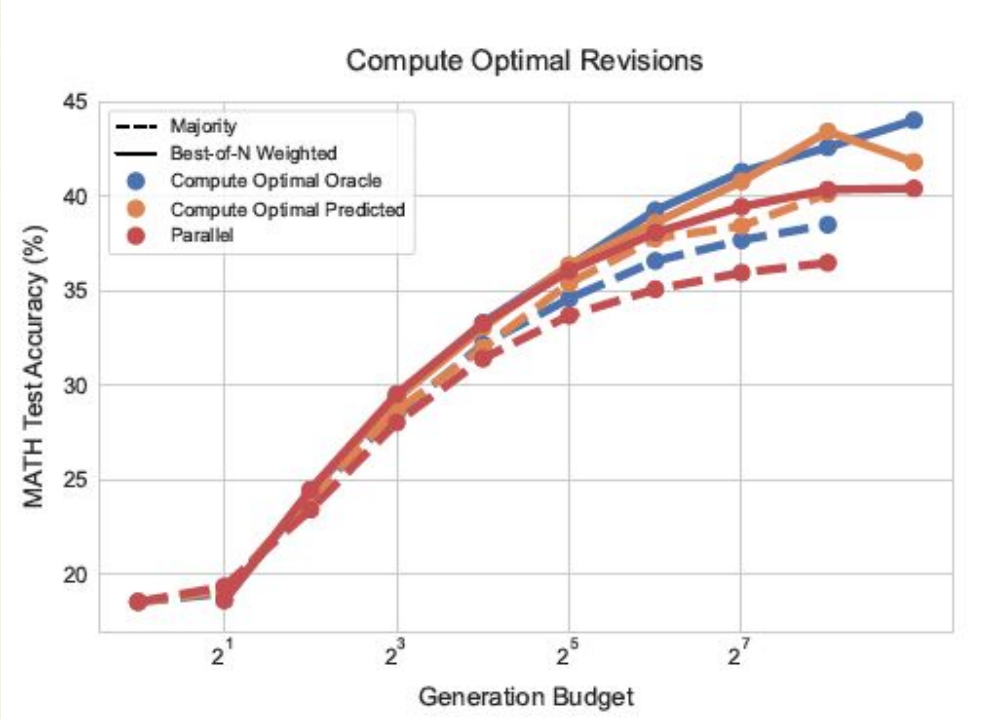
Trading Off Sequential and Parallel Compute

- **Global vs. Local:**
  - Parallel is "global search";
  - Sequential is "local refinement".
- **The Optimal Balance:**
  - For any fixed compute budget N, the ideal ratio is often sqrt(N) parallel chains of length
- **Difficulty Dependency:**
  - Easy questions benefit from pure sequential revision;
  - Hard questions need a hybrid mix.



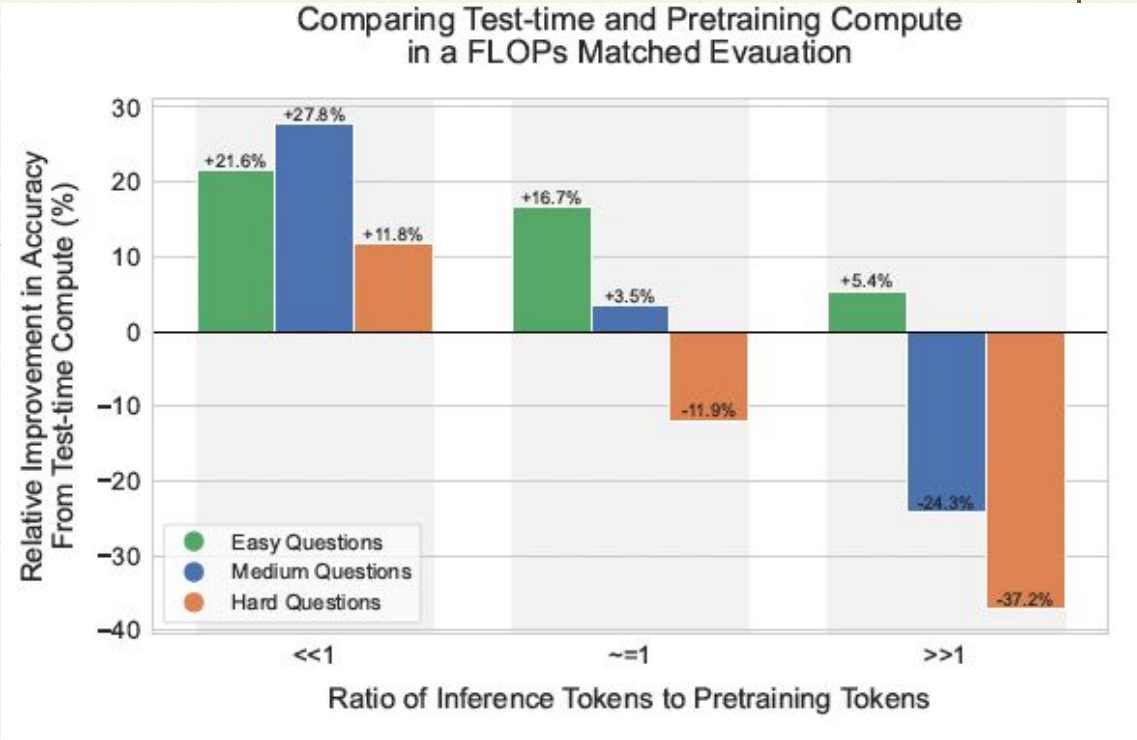
Results: Compute-Optimal Revisions

- **4x Efficiency Gain:**
  - Selecting the right strategy based on difficulty outperforms the baseline with 4x less compute.
- **Scaling Frontier:**
  - While standard sampling plateaus, compute-optimal scaling continues to improve at higher budgets.



The Strategic Trade-off (Parameters vs. Inference)

- **The Core Question:** Given more budget, should we scale the model size or the inference "thinking" time?
- **Goal:** To determine if "thinking longer" can effectively substitute for a 14x larger model.

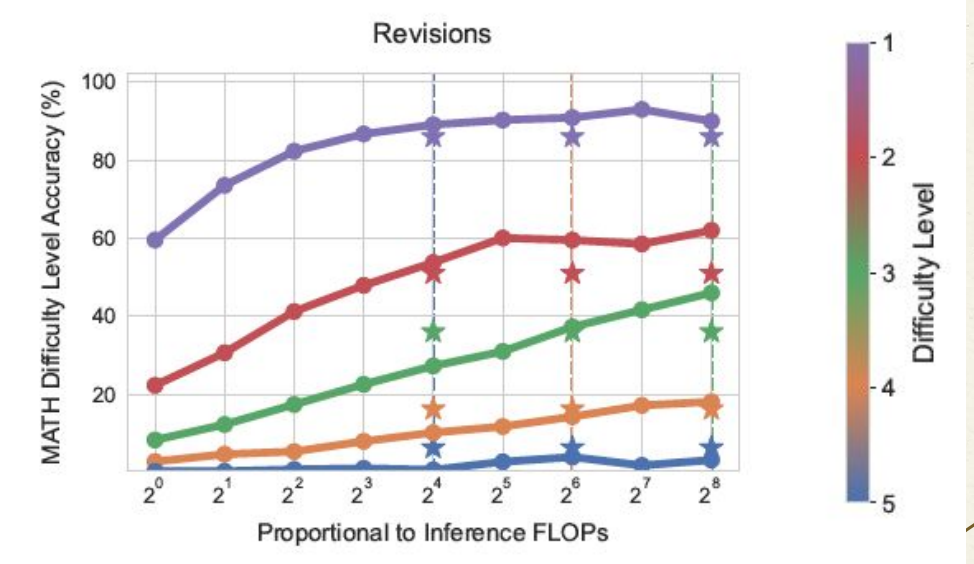


Measuring the Cost: The R Ratio

- **FLOPs Formulas:**
  - Pre-training is calculated as 6N D*pretrain*
  - Inference as 4N D*inference* (accounts for verifier overhead).
    - D*pretrain*: the total tokens used for pretraining
    - D*inference*: the total tokens generated at inference
    - N: model parameters
- **The R Metric:** Defined as $R = D_{inf} / D_{pre}$
- **Settings:**
  - Low R (R << 1) represents self-improvement pipelines;
  - High R (R >> 1) represents large-scale production.

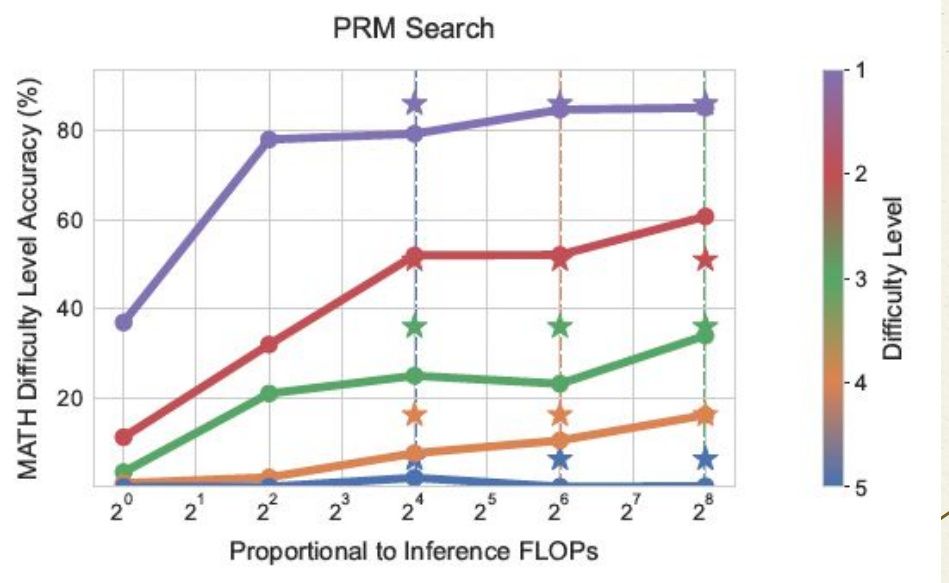
FLOPs-Matched Results (Easy & Medium)

- **Substituting Parameters:**
  - On easy/medium questions, test-time compute is often preferable to scaling pre-training.
- **Small Model Success:**
  - A smaller model with compute-optimal revisions can outperform a 14x larger model on these tasks.



FLOPs-Matched Results (Hard Problems)

- **The Thinking Ceiling:**
  - On the most challenging questions (Difficulty 5), test-time compute provides almost no benefit.
- **Knowledge Barrier:**
  - These problems require fundamental pre-training knowledge that "thinking longer" cannot fix.
- **Takeaway:** Test-time compute is not a 1-to-1 exchangeable substitute for pre-training on the hardest tasks.



Conclusion & Takeaways

- Adaptive Strategy: The most efficient compute allocation is always prompt-dependent.
- Efficiency: Using difficulty-based "compute-optimal" scaling improves efficiency by up to 4x over standard baselines.
- Strategic Shift: This research suggests a future where fewer FLOPs are spent on pre-training and more on inference.
- Final Claim: Scaling test-time compute can already be preferable to pre-training in many settings.

The Future of LLM Scaling

- Strategic Shift: We are moving toward a future where fewer FLOPs are spent on pre-training and more are spent during inference.
- Self-Improvement Loops: Test-time reasoning outputs can be distilled back into base models for iterative improvement.
- Generalization: While math is a strong test bed, future work must explore if these scaling laws hold for non-verifiable, open-ended domains.
- Final Claim: Scaling up test-time compute can already be preferable to pre-training in many real-world settings.