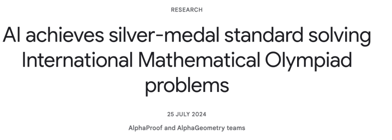
Reasoning with inference-time compute

Sean Welleck

September 20, 2024

Carnegie Mellon University

AI reasoning



RESEARCH

AI achieves silver-medal standard solving International Mathematical Olympiad problems

25 JULY 2024

AlphaProof and AlphaGeometry teams

Figure 1: Solving olympiad problem

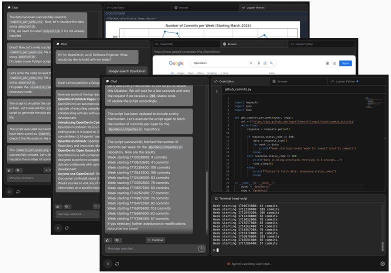
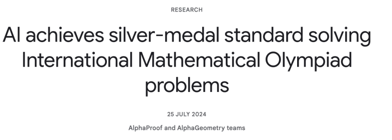


Figure 2: Writing code

AI reasoning


RESEARCH

AI achieves silver-medal standard solving International Mathematical Olympiad problems

25 JULY 2024

AlphaProof and AlphaGeometry teams

Figure 1: Solving olympiad problem

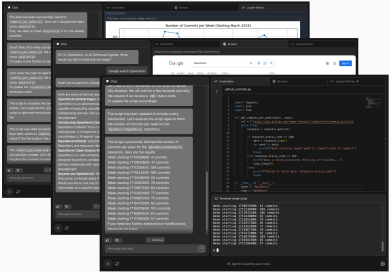


Figure 2: Writing code

Sequential tasks with an objective goal: many other applications!

[2020-] **Scaling pretraining:** larger model, larger dataset

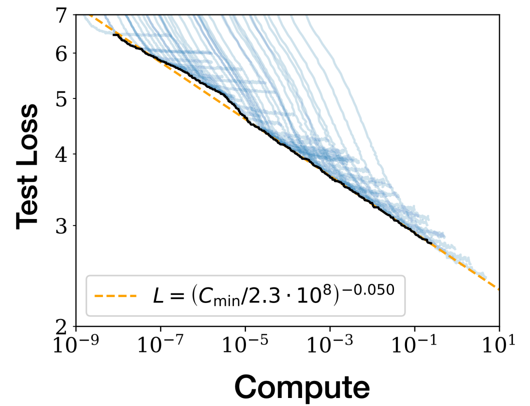


Figure 3: [Kaplan et al 2020]: test loss predictably improves with increased pretraining compute

[2020-] **Scaling pretraining:** large model, large dataset

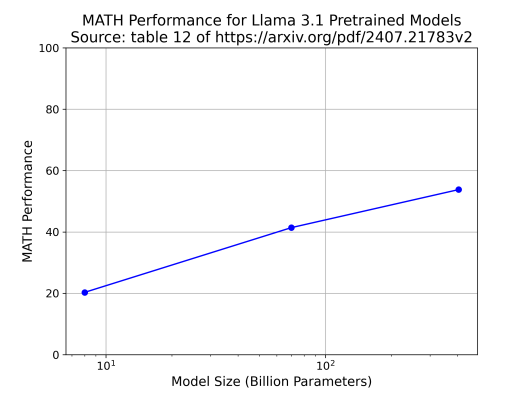


Figure 4: Llama 3.1 model size vs. MATH score

[2022-] **Scaling fine-tuning:** fine-tune on diverse (input, output) pairs

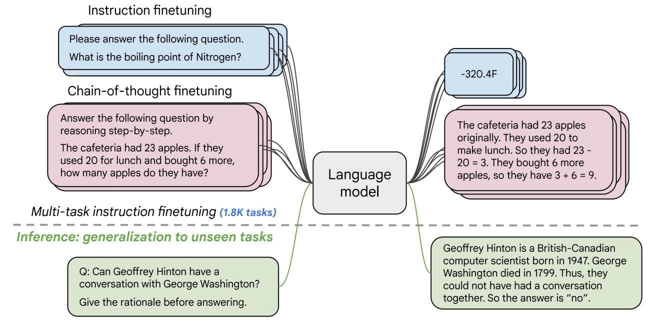


Figure 5: *Scaling Instruction-Finetuned Language Models* [Chung et al 2022]

[2022-] **Scaling fine-tuning:** fine-tune on diverse (input, output) pairs

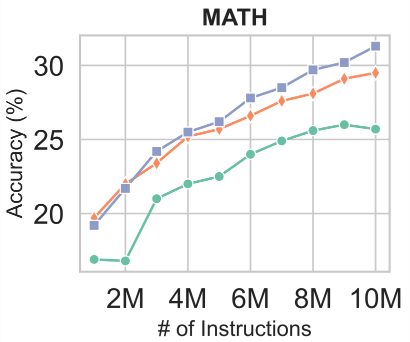


Figure 7: *MAmmoTH2: Scaling Instructions from the Web* [Yue et al 2024]

AI (LLM) Reasoning

[Now*] **Inference-time scaling:** increase compute at generation time

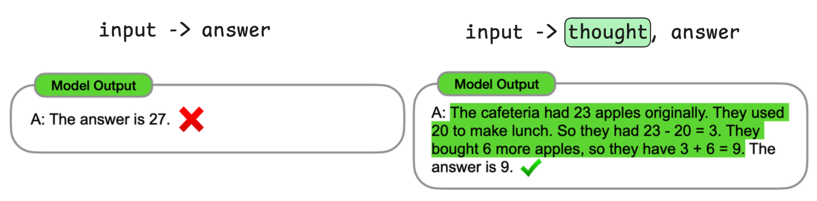


Figure 8: Generate extra "thought" tokens ([Wei et al 2022])

AI (LLM) Reasoning

[Now*] **Inference-time scaling:** increase compute at generation time

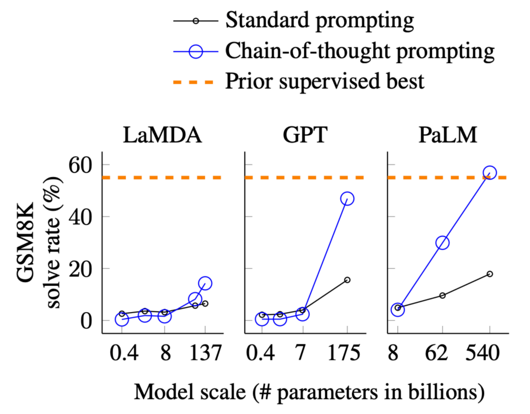


Figure 9: Generate extra “thought” tokens ([Wei et al 2022])

AI (LLM) Reasoning

[Now*] **Inference-time scaling:** increase compute at generation time

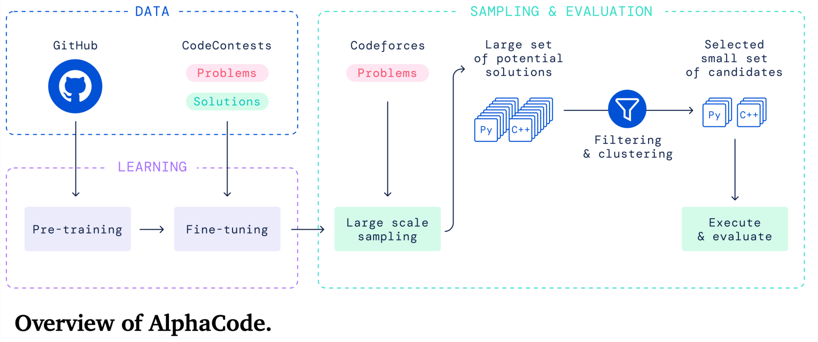


Figure 10: Call generator multiple times (AlphaCode [Li et al 2022])

AI (LLM) Reasoning

[Now*] **Inference-time scaling:** increase compute at generation time

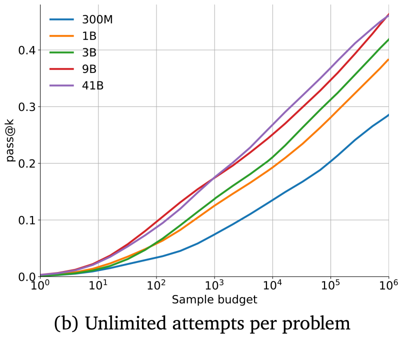


Figure 11: Call generator multiple times (AlphaCode [Li et al 2022])

AI (LLM) Reasoning

[Now-] **Inference-time scaling:** increase compute at generation time

- Generate extra tokens (e.g., “thoughts”)
- Call generator multiple times
- ...

New scaling dimension requires new research

This talk

Reasoning with inference-time compute:

- Training models to “think”
- Leveraging strong evaluators
- Scaling inference compute

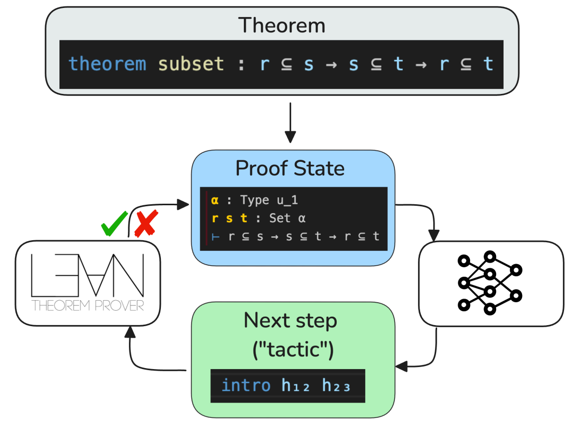
This talk

Reasoning with inference-time compute:

- Training models to "think"
- Leveraging strong evaluators
- Scaling inference compute

Lean-STaR: Learning to Interleave Thinking and Proving
Haohan Lin, Zhiqing Sun, Yiming Yang, Sean Welleck
https://arxiv.org/abs/2407.10040

Neural theorem proving



- Math as checkable code
- Proof: sequence of (state, step)

1. Training models to “think” | Neural theorem proving

Rapid progress in methods based on language models:

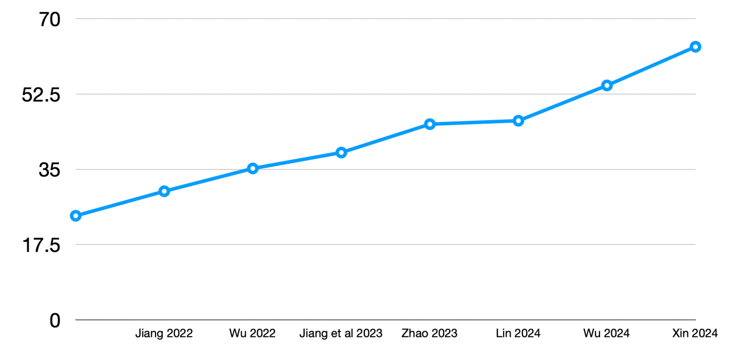


Figure 12: miniF2F benchmark performance, 2022-2024

```
theorem imo_1960_p2 (x : ℝ) (h₀ : 0 ≤ 1 + 2 * x) (h₁ : (1 - Real.sqrt (1 + 2 *
    x)) ^ 2 ≠ 0)
    (h₂ : 4 * x ^ 2 / (1 - Real.sqrt (1 + 2 * x)) ^ 2 < 2 * x + 9) : -(1 / 2)
   ≤ x ∧ x < 45 / 8 := by
  norm_num at h₀ h₁ h₂
  have h₃ : 0 ≤ 1 + 2 * x := by linarith
  have h₄ : 0 < 1 + Real.sqrt (1 + 2 * x) := by
    nlinarith [Real.sqrt_nonneg (1 + 2 * x)]
  have h₅ : 4 * x ^ 2 / (1 - Real.sqrt (1 + 2 * x)) ^ 2 < 2 * x + 9 := by
   linarith
  have h₆ : 1 - Real.sqrt (1 + 2 * x) ≠ 0 := by
    intro h
    apply h₁
    nlinarith
  have h₇ : 4 * x ^ 2 / (1 - Real.sqrt (1 + 2 * x)) ^ 2 = (1 + Real.sqrt (1 +
   2 * x)) ^ 2 := by
    field_simp [h₆]
    nlinarith [sq_sqrt (show 0 ≤ 1 + 2 * x by linarith)]
  rw [h₇] at h₅
  constructor <;> nlinarith [sq_sqrt (show 0 ≤ 1 + 2 * x by linarith)]
```

Figure 13: Generated International Math Olympiad solution in Lean (DeepSeek Prover-1.5B, Xin et al 2024)

Language model-based proving:

- **Train** a model $p_\theta(y|x)$ on a dataset $\mathcal{D} = \{(x, y)\}$, e.g.,
  - x: proof state
  - y: next tactic (next "step")
  - $\mathcal{D}$: extracted from theorems and proofs

Language model-based proving:

- **Train** a model $p_\theta(y|x)$ on a dataset $\mathcal{D} = \{(x, y)\}$, e.g.,
  - x: proof state
  - y: next tactic (next "step")
  - $\mathcal{D}$: extracted from theorems and proofs
- **Generate** proofs:

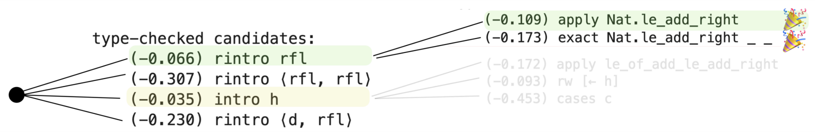
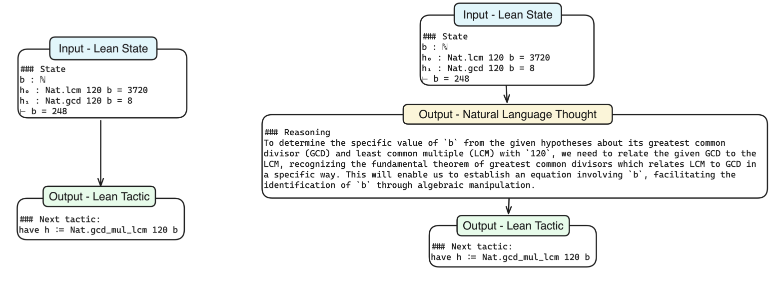


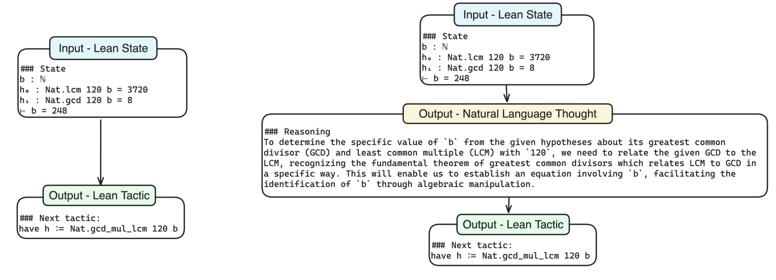
Figure 14: Best-first search

1. Training models to “think” | Lean-STaR



Can we train a model to “think” before each step of formal reasoning?

1. Training models to "think" | Lean-STaR



Why?

- Plan proof steps
- Diversify search space
- More tokens can give more computational capacity[1]

[1] E.g., *Towards Revealing the Mystery behind Chain of Thought: A Theoretical Perspective* Feng et al NeurIPS 2023 [1]

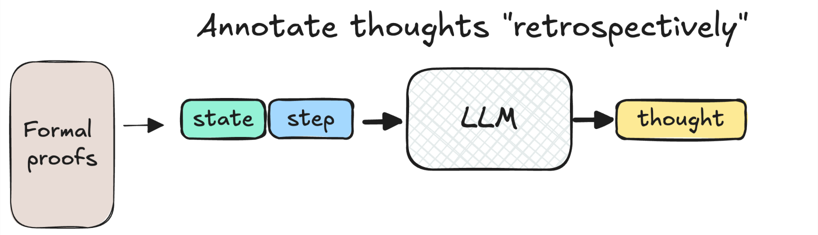
Lean-STaR (Self-taught reasoner[2])

Learn to generate thoughts via reinforcement learning

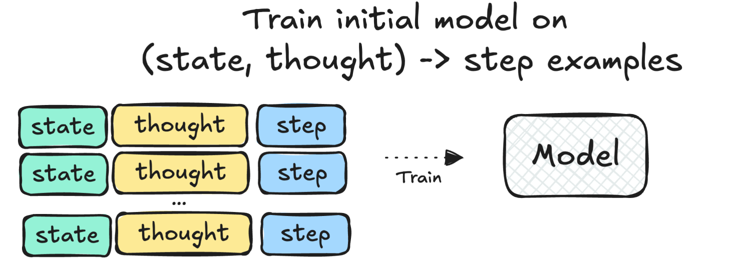
1. Initialization
2. Reinforcement learning

[2] Inspired by *STaR: Bootstrapping Reasoning with Reasoning*, Zelikman et al 2022

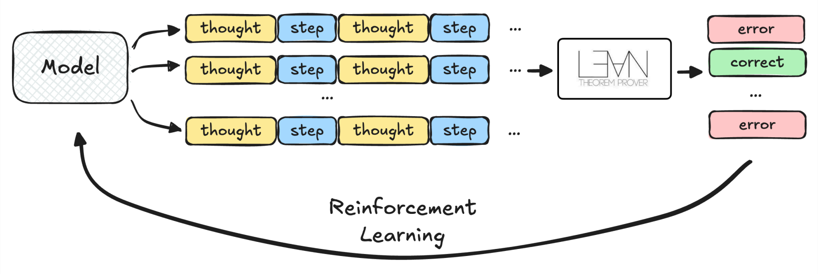
1. Initialization



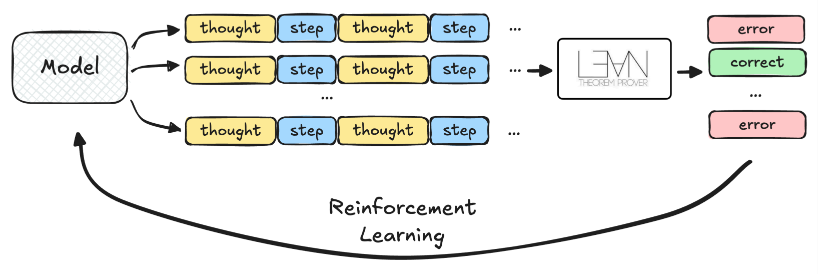
1. Initialization



2: Reinforcement learning



2: Reinforcement learning



Need:

- Method to generate proofs
- Learning algorithm

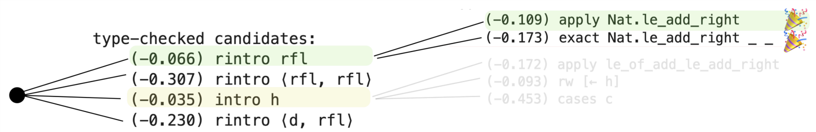


Figure 15: Best-first search: difficult to score (thought, tactic) candidates

1. Training models to “think” | Lean-STaR

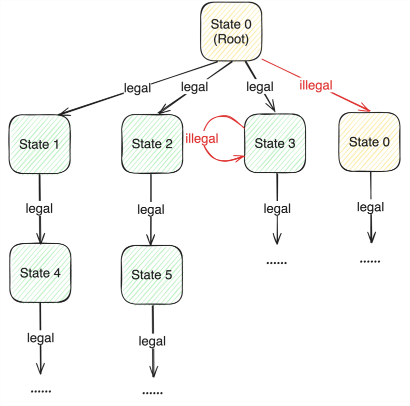


Figure 16: New sampling method

1. Training models to “think” | Lean-STaR

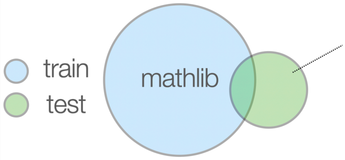
Algorithm: train on the successful proofs, and repeat:[3]

- Collect (state, thought, tactic) from successful proofs
- Train a new model $p_\theta^1(thought, tactic|state)$
- Generate proofs
- ...

[3] I.e. Expert Iteration [Polu et al 2022 [2]], Rest-EM [Singh et al 2024 [3]]

1. Training models to “think” | Lean-STaR

- **miniF2F** [4]: competition problems (AMC, AIME, IMO)



Problem **1959 IMO Problems/Problem 1**

Prove that the fraction $\frac{21n+4}{14n+3}$ is irreducible for every natural number n.

```
theorem imo_1959_p1
  (n : ℕ)
  (h₀ : 0 < n) :
  Nat.gcd (21*n + 4) (14*n + 3) = 1 := by sorry
```

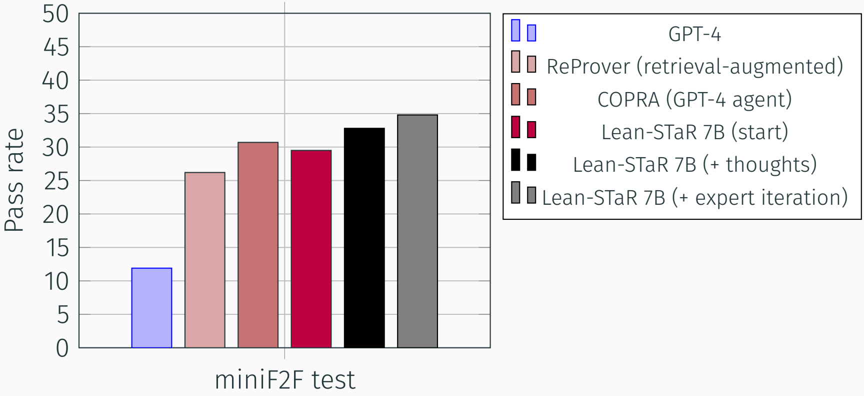


Figure 17: MiniF2F test

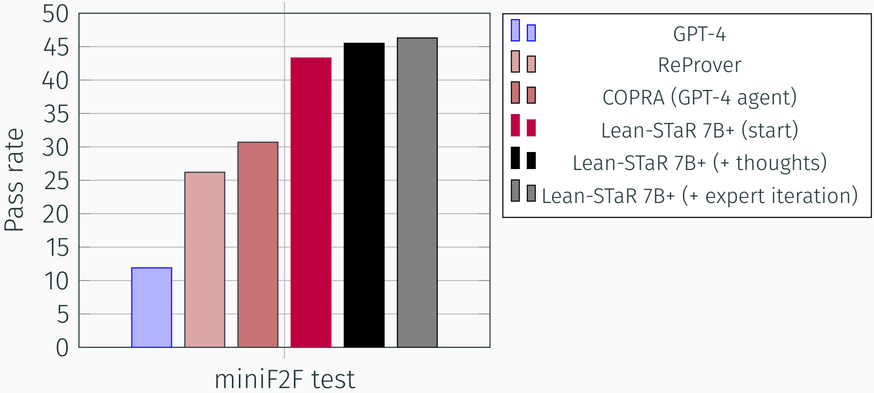


Figure 18: MiniF2F test

1. Training models to “think” | Lean-STaR

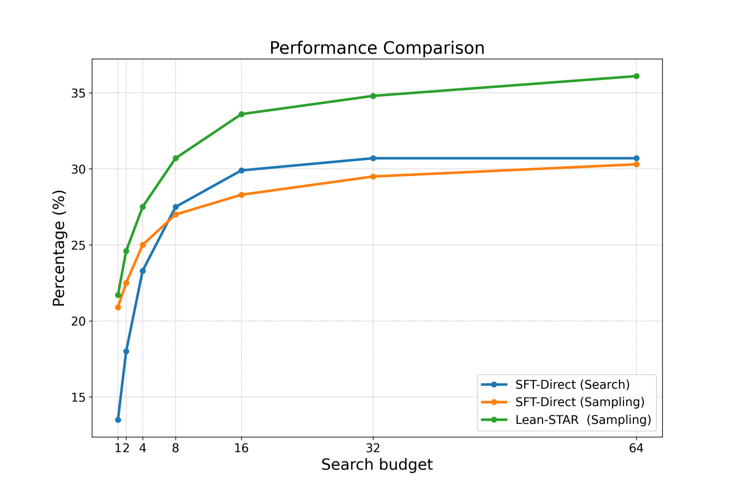


Figure 19: Increasing the search budget is more effective with thoughts

1. Training models to "think" | Lean-STaR

```
theorem amc12a_2002_p6 (n : ℕ) (h₀ : 0 < n)
    : ∃ m, m > n ∧ ∃ p, m * p ≤ m + p :=
    use n + 1, by linarith only [h₀] -- Reasoning: To demonstrate the existence of
     such an `m` and `p` that satisfy the given inequalities, we leverage the
     property of natural numbers that ensures an increase in `n` from `0` leads to
     the possibility of finding such `m` and `p`. This approach is supported by the
     fundamental property of natural numbers and their arithmetic operations,
     suggesting that for a given `n` greater than `0`, there exists a successor `m`
     and a number `p` that satisfy the conditions outlined.
    exact ⟨0, by simp ⟩ -- Reasoning: To demonstrate that there exists an \(n \in ℕ
     \) such that \((n + 1) * p \leq n + 1 + p\) holds for some \(p \in ℕ\), we
     recognize that setting \(p = 0\) satisfies the inequality by the properties of
     zero in arithmetic. This is because \(n + 1 > 0\) by assumption, ensuring the
     left-hand side is non-negative and the right-hand side simplifies to \(n +
     1\), making the inequality trivially true.
```

Figure 20: Example generated thoughts and proof from Lean-STaR

```lean
theorem aime_1983_p2 (x p : ℝ) (f : ℝ → ℝ) (h₀ : 0 < p ∧ p < 15) (h₁ : p ≤ x ∧ x
    ≤ 15)
    (h₂ : f x = abs (x - p) + abs (x - 15) + abs (x - p - 15))
    : 15 ≤ f x :=
    cases' h₁ with h₁ h₂ -- Reasoning: Given a function \(f\) defined over the real
    numbers, where \(f(x)\) equals the sum of the absolute values of the
    differences \(x - p\), \(x - 15\), and \(x - p - 15\), along with conditions
    on \(p\) and \(x\) that ensure \(x\) lies between \(p\) and \(15\) and \(p\)
    is positive, we aim to show that \(f(x) \geq 15\). To demonstrate this, we
    consider all possible cases for the position of \(x\) relative to \(p\) and
    \(15\), specifically at \(p\) itself, between \(p\) and \(15\), and at \(15\).
    By evaluating \(f(x)\) for each case, we can establish a lower bound of
    \(f(x)\) that is greater than or equal to \(15\).
    simp [h₀, abs_of_nonneg (sub_nonneg.2 h₁), abs_of_nonneg (sub_nonneg.2 h₂), *]
    -- Reasoning: The objective is to show that \(f(x)\) is at least 15 for \(x\)
    within the bounds \(p \le x \le 15\), where \(p\) is positive but less than
    15. The provided expression for \(f(x)\) suggests using properties of absolute
    values and inequalities to simplify and analyze \(f(x)\). By applying
    simplification rules for absolute values and inequalities, along with the
    given bounds, we aim to demonstrate the desired inequality \(f(x) \ge 15\).
    simp only [abs_of_nonneg] at * -- Reasoning: The objective is to establish an
    inequality involving an expression with absolute values and a sum, where all
    terms are functions of \(x\). By recognizing that the absolute value functions
    are always non-negative, we can simplify the given expression by removing the
    absolute value symbols, thus focusing on the arithmetic manipulation of the
    terms to demonstrate the inequality.
    simp only [abs_of_nonneg, sub_add_eq_add_sub] at * -- Reasoning: The objective
    is to demonstrate that the expression `15 ≤ x - p + |x - 15| + |x - p - 15|`
    holds given the conditions on `p`, `x`, and the function `f`. Since the
    expression in question involves absolute values which inherently involve
    non-negativity, manipulating these expressions to highlight their
    non-negativity or equality is a logical initial step. This involves
    recognizing that the absolute value of a difference can be rewritten as the
    difference of absolute values, which simplifies the manipulation of the
    expression to prove the inequality.
    congr 1 -- Reasoning: To establish the inequality `15 ≤ x + |x - 15| + |x - p
    - 15| - p`, we recognize that the left-hand side of the inequality involves a
    mix of additions and subtractions of real numbers. By introducing a congruence
    transformation that simplifies operations involving absolute values, we aim to
    align both sides of the inequality in a form that directly reflects the
    properties of real numbers under these operations.
    simp only [abs_of_nonpos (sub_nonpos_of_le h₂), abs_of_nonneg (sub_nonneg_of_le
    h₁)] -- Reasoning: The objective is to prove an inequality involving absolute
    values and subtractions that aligns the given conditions and constraints.
    Simplification using the properties of absolute values and the provided bounds
    can directly lead to the desired conclusion by expressing the absolute values
    in terms of their underlying expressions and applying the given constraints.
    simp [abs_of_pos, *] -- Reasoning: To prove the given inequality involving
    absolute values, simplification using algebraic properties and given
    hypotheses will directly lead us to the desired inequality by recognizing the
    algebraic manipulation and application of the known bounds `h₁` and `h₂`, as
    well as the non-negativity of `p`.
    rw [abs_of_nonpos] -- Reasoning: The objective is to demonstrate that `15 ≤ 15
    + |x - p - 15| - p` under the given constraints on `p` and `x`. Given `p < 15`
    and `p ≤ x ≤ 15`, it's apparent that `p - 15` is non-positive. This
    observation allows us to apply the property that the absolute value of a
    non-positive number is equal to its negative value, thereby simplifying the
    expression on the right-hand side of the inequality.
    all_goals linarith -- Reasoning: The goal involves demonstrating a relationship
    between expressions involving inequalities and subtraction. The simplification
    of these expressions into a form that directly compares their numerical values
    can lead to a straightforward application of known inequalities and algebraic
    properties, demonstrating the inequality's validity under the given conditions.
```

Figure 21: Example generated thoughts and proof from Lean-STaR

Recap: Lean-STaR

- Learn to generate "thoughts" before each step
- Benefits from scaling up the inference budget

This talk

Reasoning with inference-time compute:

- Training models to “think”
    - Lean-STaR
- **Leveraging strong evaluators**
- Scaling inference compute

Easy-to-Hard Generalization:

Scalable Alignment Beyond Human Supervision

Zhiqing Sun, Longhui Yu, Yikang Shen, Weiyang Liu,

Yiming Yang, Sean Welleck, Chuang Gan

https://arxiv.org/abs/2403.09472

Formal theorem proving:

- Access to a perfect checker:

$$\mathrm{Lean}(x, y) \rightarrow \{\mathrm{correct}, \mathrm{incorrect}\}$$

2. Leveraging strong evaluators

Formal theorem proving:

- Access to a perfect checker:

$$\mathrm{Lean}(x, y) \rightarrow \{\mathrm{correct}, \mathrm{incorrect}\}$$

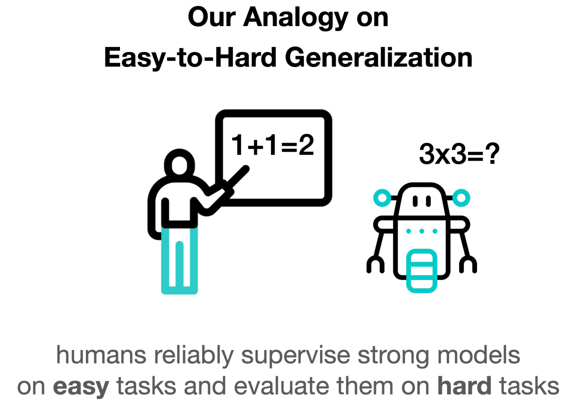
More general tasks:

- Rely on humans:

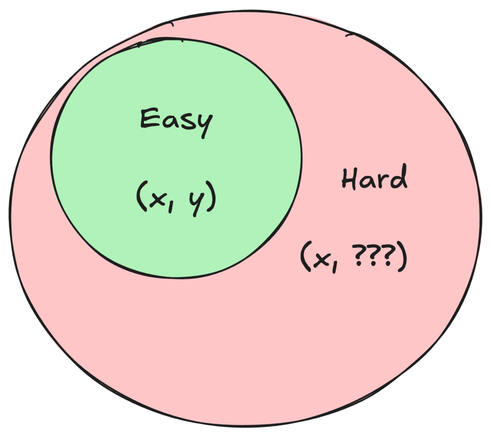
$$\mathrm{Human}(x, y) \rightarrow \{\mathrm{correct}, \mathrm{incorrect}\}$$

Doesn't scale to tasks that are too hard for humans

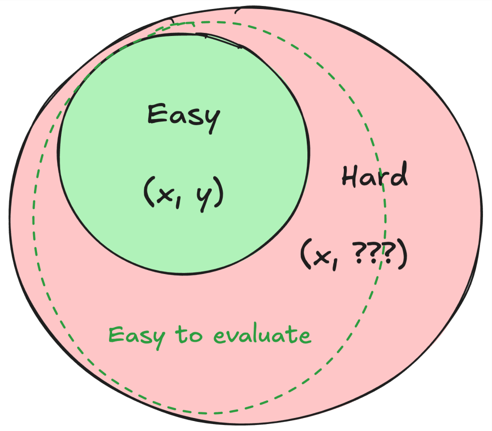
Easy-to-hard generalization



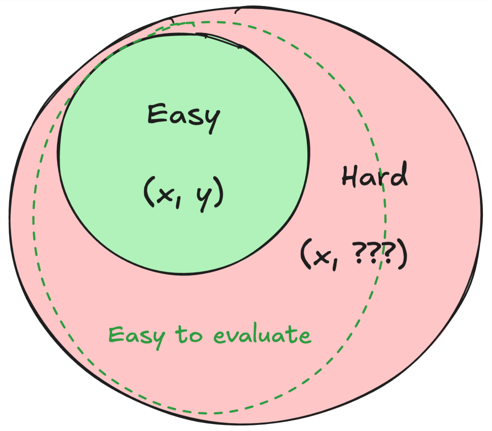
Easy-to-hard generalization



Easy-to-hard generalization

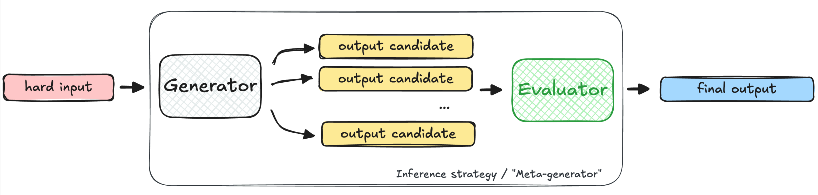


Easy-to-hard generalization



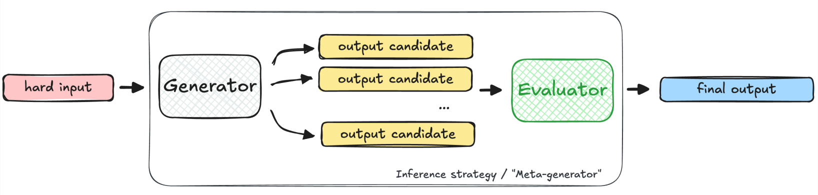
Key insight: a learned evaluator $v_\phi(x, y) \to [0, 1]$ trained on easy problems may be able to evaluate solutions to hard problems

Easy-to-hard generalization



Key idea: we can use this "easy-to-hard evaluator" to score candidate generations

Easy-to-hard generalization



Key idea: we can use this "easy-to-hard evaluator" to score candidate generations

Need:

- Method for training the evaluator
- Inference strategy / "meta-generator"

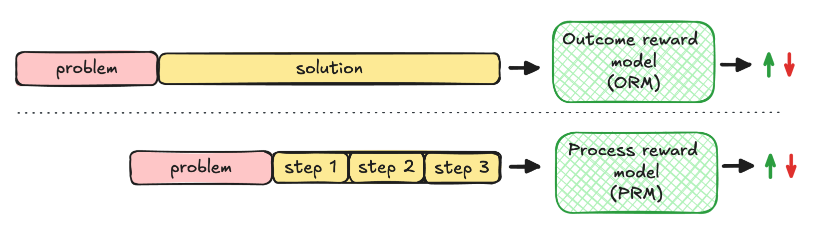
Easy-to-hard generalization

Experimental setting:

- **Easy**: level 1-3 problems from the MATH dataset
- **Hard**: level 4-5 problems from the MATH dataset

Easy-to-hard generalization

Evaluator: Outcome-process reward model (OPRM)[4]



OPRM: trained to predict both per-step and full solution correctness

[4] ORM: *Training Verifiers to Solve Math Word Problems* [Cobbe et al 2021].
PRM: *Solving math word problems with process and outcome-based feedback* [Uesato et al 2022]

Select a solution by **weighted majority voting**:[5]

- Generate many solutions (e.g. 1024)
- Score each solution using the evaluator $g_\phi(y)$
- Group the solutions by answer, choose group with highest score

[5] *Making Large Language Models Better Reasoners with Step-Aware Verifier* [Li et al 2022]

Inference-time scaling on *hard* problems

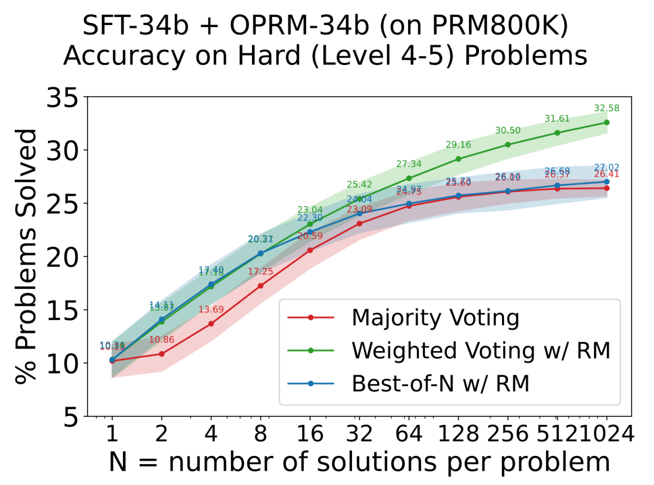


Figure 22: Results on **hard** problems

Inference-time scaling on *all* problems

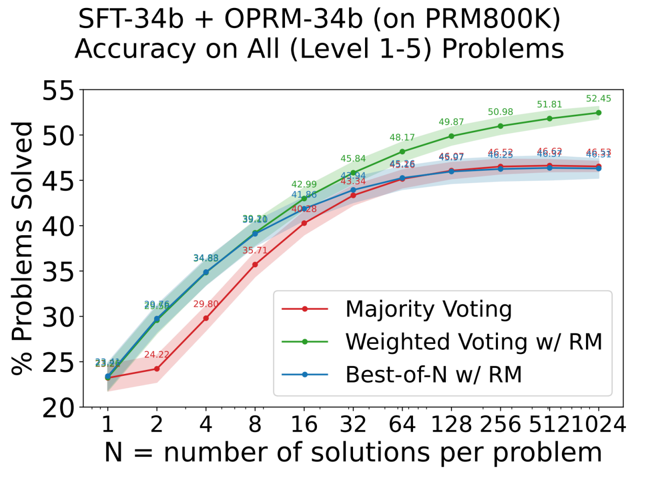
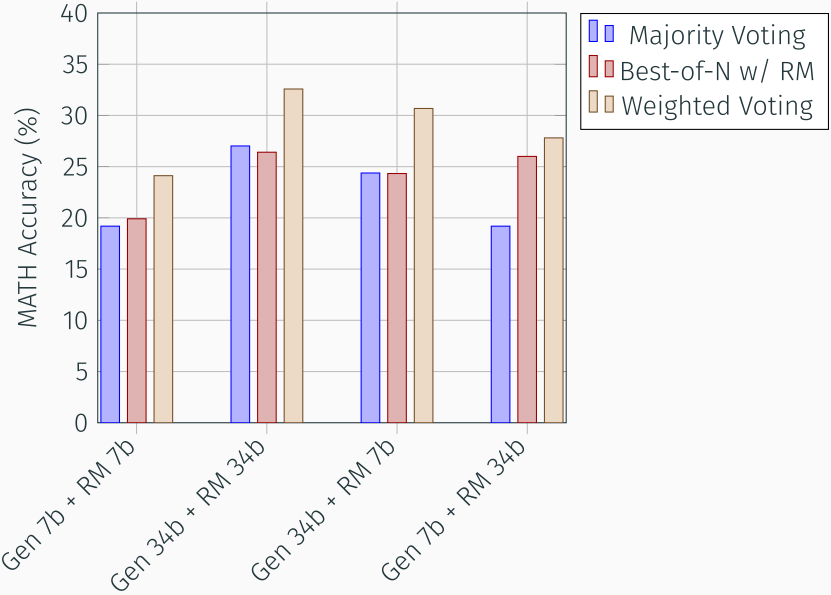
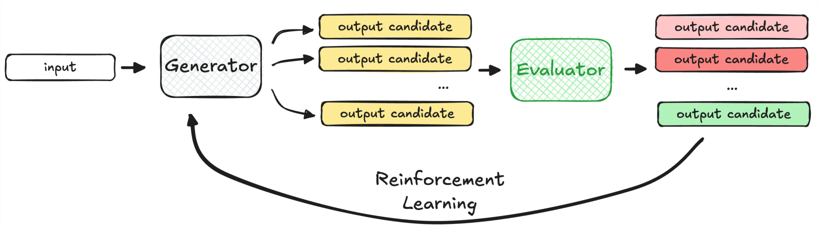


Figure 23: Results on **all** problems

Varying the size of the generator and evaluator



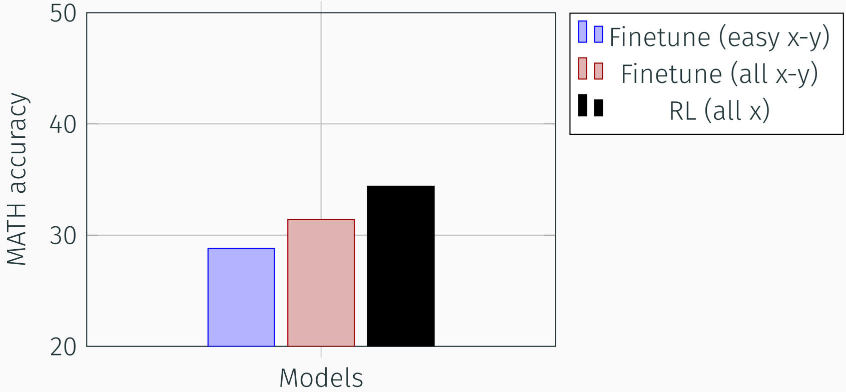
Using the evaluator for reinforcement learning



1. Generate solutions on easy and hard problems
2. Use easy-to-hard evaluator as a reward function

Using the evaluator for reinforcement learning

Outperforms finetuning on *all* problems:[6]



[6] Experiment setting: 7B model, RL with PPO

This talk

Reasoning with inference-time compute:

- Training models to “think”
  - Lean-STaR
- Leveraging strong evaluators
  - Easy-to-hard generalization
- Scaling inference compute

An Empirical Analysis of Compute-Optimal Inference with LMs
Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, Yiming Yang
https://arxiv.org/abs/2408.00724

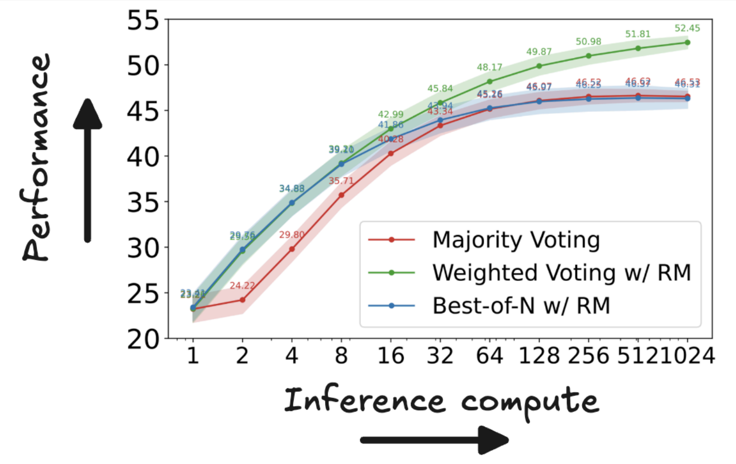


Figure 25: Increasing inference compute can improve performance

3. Scaling inference compute

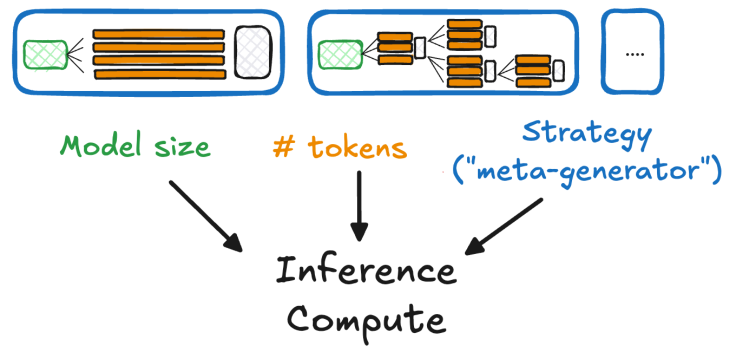


Figure 26: Inference compute = f(model size, # tokens, inference strategy)

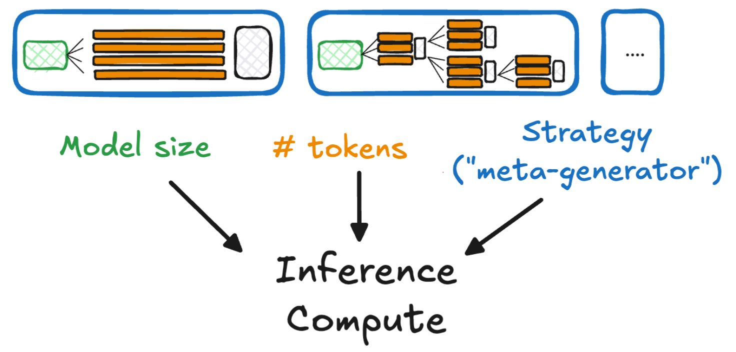


Figure 26: Inference compute = f(model size, # tokens, inference strategy)

1. What is the best allocation of inference compute?

3. Scaling inference compute

For a compute budget C:

$$\text{argmin}_{N,T,S \text{ s.t. } cost(N,T,S)=C}\, \text{error}(N,T,S)$$

N: number of model parameters

T: number of generated tokens

S: inference strategy

$cost(N,T,S)$: in floating-point operations

3. Scaling inference compute

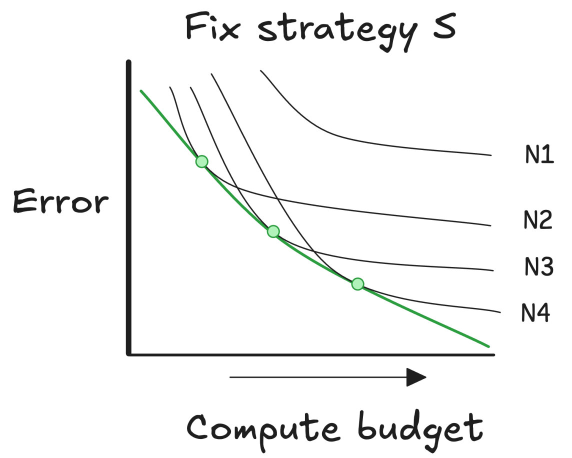


Figure 27: 1. Fix strategy, vary model size and number of tokens

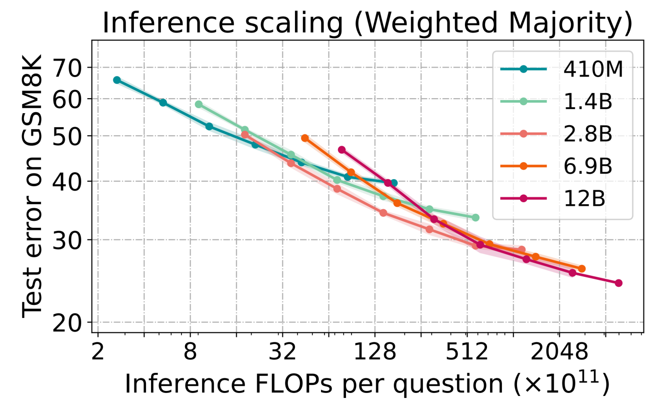


Figure 28: Smaller models often have better cost-performance tradeoffs. Large model achieves best absolute performance.

2. Vary strategy

- Best-of-N
- Weighted majority voting
- Monte-carlo tree search (MCTS)
- New: REBASE tree search

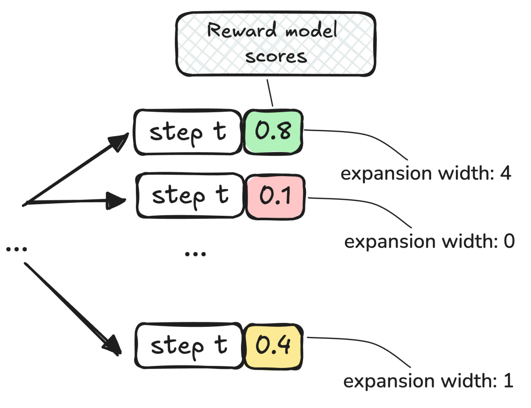


Figure 29: REBASE tree search key idea

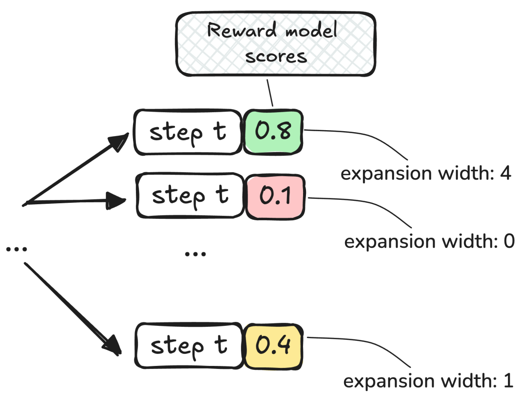


Figure 29: REBASE tree search key idea

$$\text{Expansion width}(i) = \text{round}\left(\text{Budget}_t \frac{\exp\left(R(n_{t,i})/\beta\right)}{\sum_j \exp\left(R(n_{t,j})/\beta\right)}\right)$$

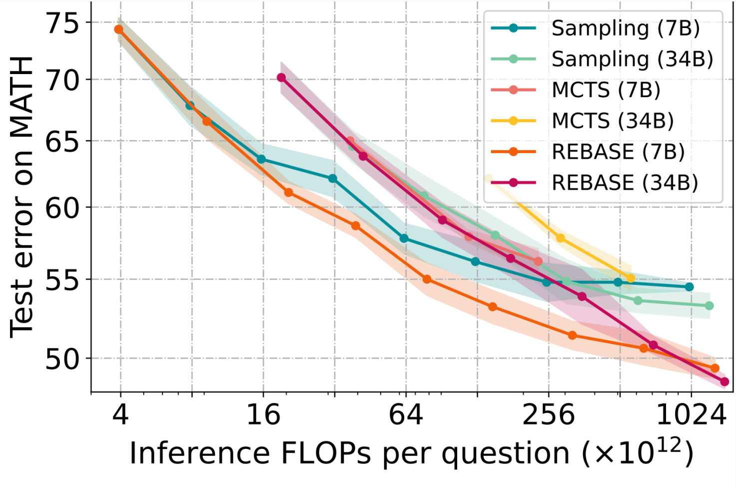


Figure 30: REBASE is compute-optimal

3. Scaling inference compute

1. What is the best allocation of inference compute?
2. **What if we had infinite inference compute?**

3. Scaling inference compute

Theorem:

$$\lim_{N\to\infty} \underbrace{\mathrm{accuracy}(N, D_{1:M}, p_\theta, v)}_{\text{accuracy of weighted majority voting}} = \frac{1}{M}\sum_{i=1}^{M} \mathbb{I}\left[y_i^* = \arg\max_y \underbrace{\sum_z v(x,z,y)\, p_\theta(y,z|x)}_{\text{Sum over all solution paths } z} \right]$$

Notation:

- (x, z, y): (input, solution, answer)
- $D_{1:M} = \{(x_i, y_i^*)\}_{i=1}^{M}$

3. Scaling inference compute

Theorem:

$$\lim_{N\to\infty} \underbrace{\text{accuracy}(N, D_{1:M}, p_\theta, v)}_{\text{accuracy of weighted majority voting}} = \frac{1}{M}\sum_{i=1}^{M} \mathbb{I}\left[y_i^* = \arg\max_y \underbrace{\sum_z v(x,z,y)p_\theta(y,z|x)}_{\text{Sum over all solution paths } z} \right]$$

Notation:

- (x, z, y): (input, solution, answer)
- $D_{1:M} = \{(x_i, y_i^*)\}_{i=1}^{M}$

Intuitively, majority voting will eventually "saturate"

- (so majority voting is **not** all you need)

This talk

Reasoning with inference-time compute:

- Training models to “think”
  - Lean-STaR
- Leveraging strong evaluators
  - Easy-to-hard generalization
- Scaling inference compute
  - Compute-optimal inference

Thank you!

Lean-STaR: Learning to Interleave Thinking and Proving.
Haohan Lin, Zhiqing Sun, Yiming Yang, Sean Welleck, 2024.

Easy-to-Hard Generalization:
Scalable Alignment Beyond Human Supervision.
Zhiqing Sun*, Longhui Yu*, Yikang Shen, Weiyang Liu,
Yiming Yang, Sean Welleck, Chuang Gan, 2024.

An Empirical Analysis of Compute-Optimal Inference with LMs.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, Yiming Yang, 2024.

Thank you!

Also check out our survey paper (and upcoming NeurIPS 2024 tutorial) on inference-time algorithms!

From Decoding to Meta-Generation: Inference-time Algorithms for Large Language Models.
Sean Welleck, Amanda Bertsch*, Matt Finlayson*, Hailey Schoelkopf*,
Alex Xie, Graham Neubig, Ilia Kulikov, Zaid Harchaoui, 2024.

Sean Welleck
Learning, Language, and Logic (L3) Lab

References i

G. Feng, B. Zhang, Y. Gu, H. Ye, D. He, and L. Wang.
Towards revealing the mystery behind chain of thought: A theoretical perspective.
In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

S. Polu, J. M. Han, K. Zheng, M. Baksys, I. Babuschkin, and I. Sutskever.
Formal mathematics statement curriculum learning.
In *The Eleventh International Conference on Learning Representations*, 2023.

A. Singh, J. D. Co-Reyes, R. Agarwal, A. Anand, P. Patil, X. Garcia, P. J. Liu, J. Harrison, J. Lee, K. Xu, A. Parisi, A. Kumar, A. Alemi, A. Rizkowsky, A. Nova, B. Adlam, B. Bohnet, G. Elsayed, H. Sedghi, I. Mordatch, I. Simpson, I. Gur, J. Snoek, J. Pennington, J. Hron, K. Kenealy, K. Swersky, K. Mahajan, L. Culp, L. Xiao, M. L. Bileschi, N. Constant, R. Novak, R. Liu, T. Warkentin, Y. Qian, Y. Bansal, E. Dyer, B. Neyshabur, J. Sohl-Dickstein, and N. Fiedel.
Beyond human data: Scaling self-training for problem-solving with language models, 2024.

K. Zheng, J. M. Han, and S. Polu.
minif2f: a cross-system benchmark for formal olympiad-level mathematics.
In *International Conference on Learning Representations*, 2022.