The *Key* Ingredients for Scaling Test-Time Compute

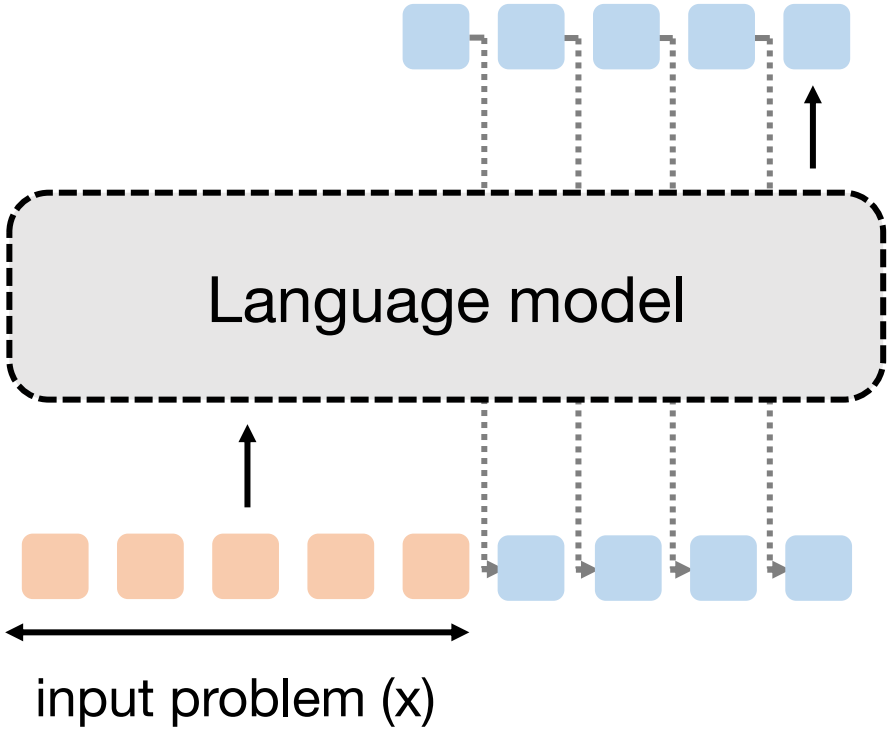
(and what is still missing)

Aviral Kumar

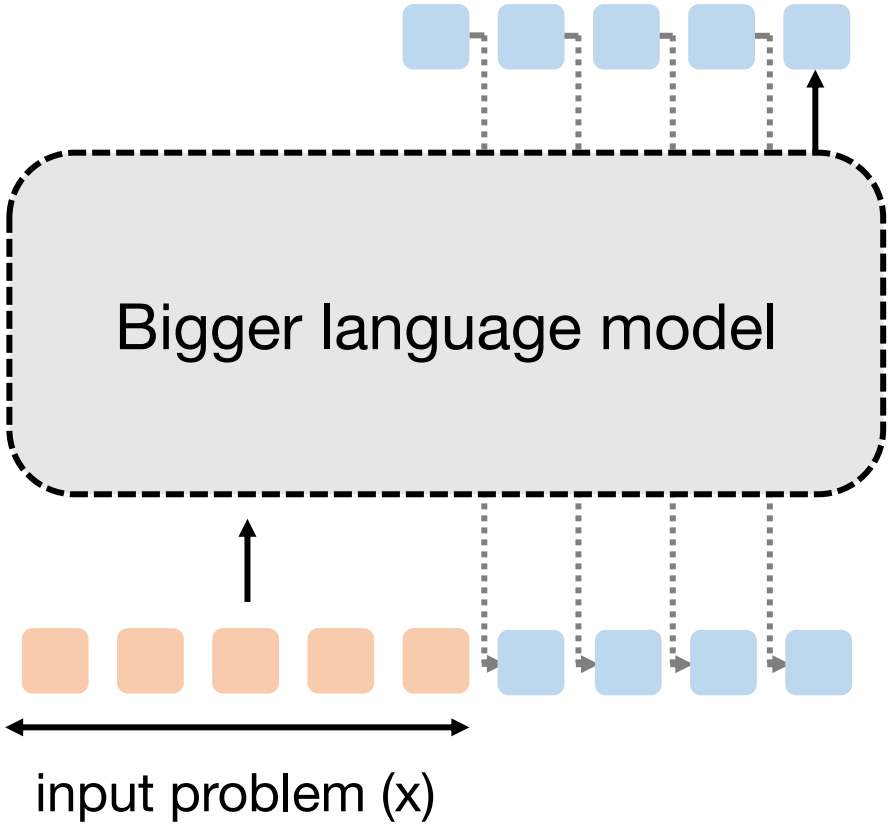
Carnegie Mellon University

School of Computer Science

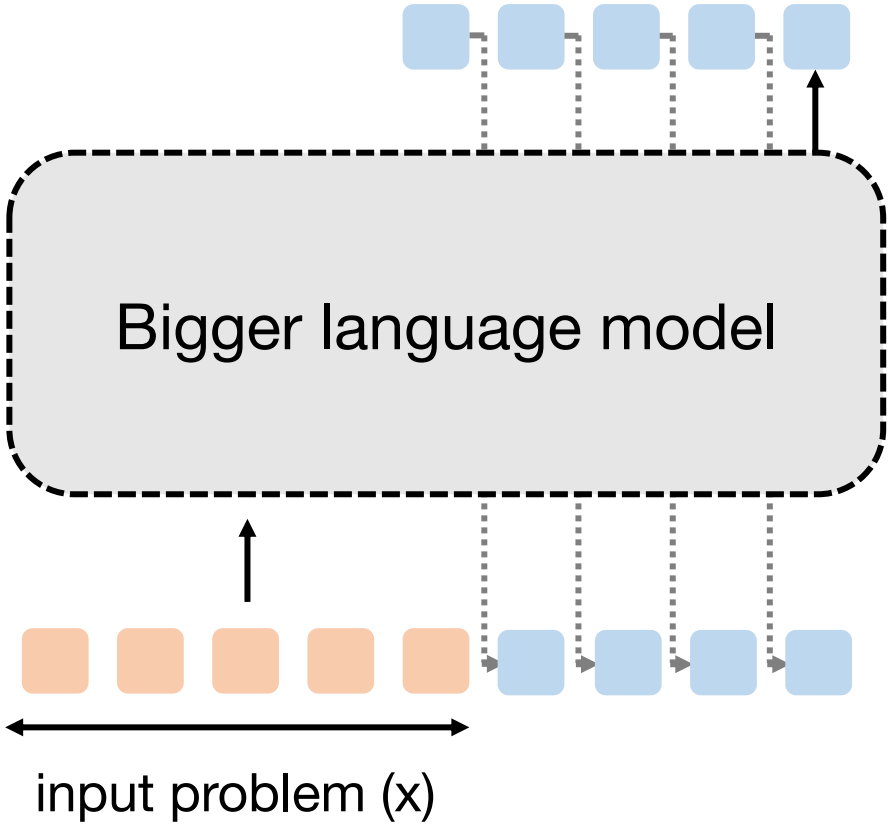
Test-Time Scaling for Large Language Models

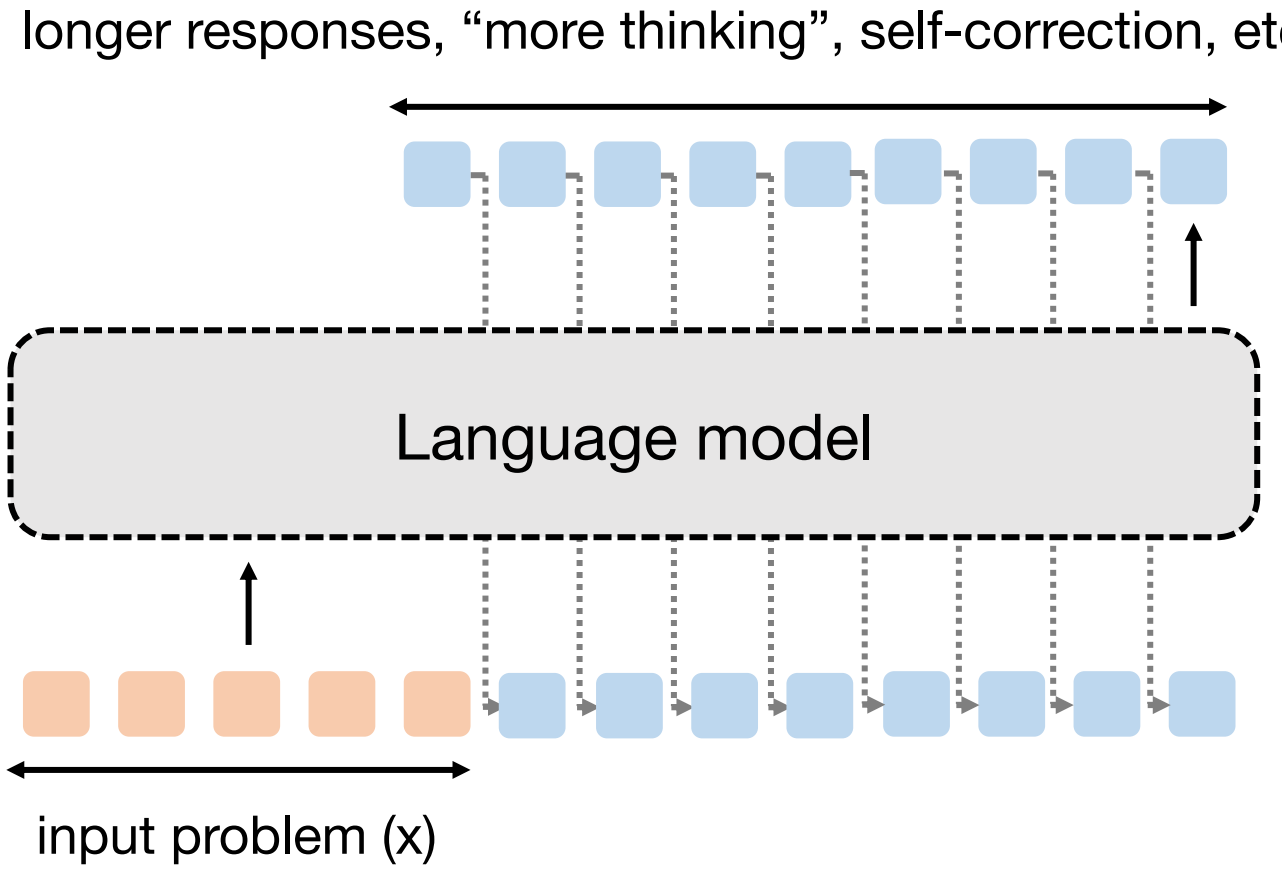


Test-Time Scaling for Large Language Models



Test-Time Scaling for Large Language Models



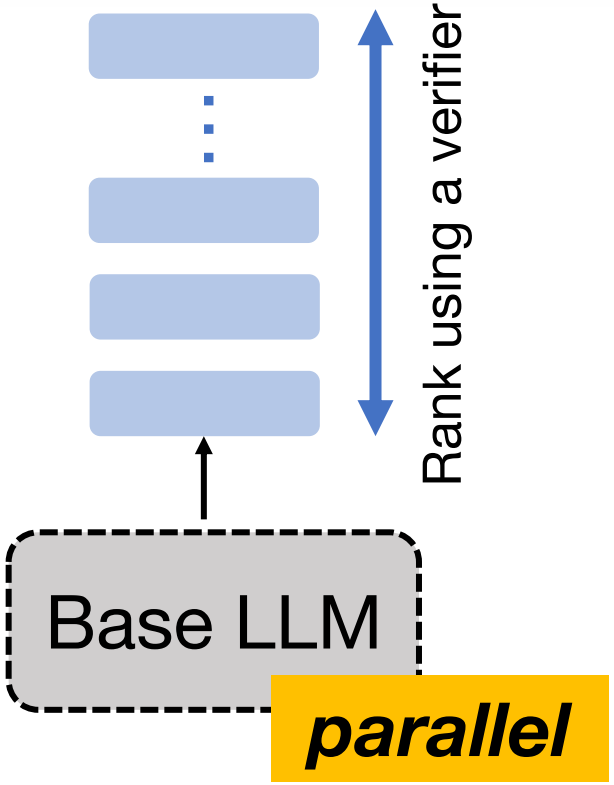


Finetuning LLMs is Critical for Test-Time Scaling

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

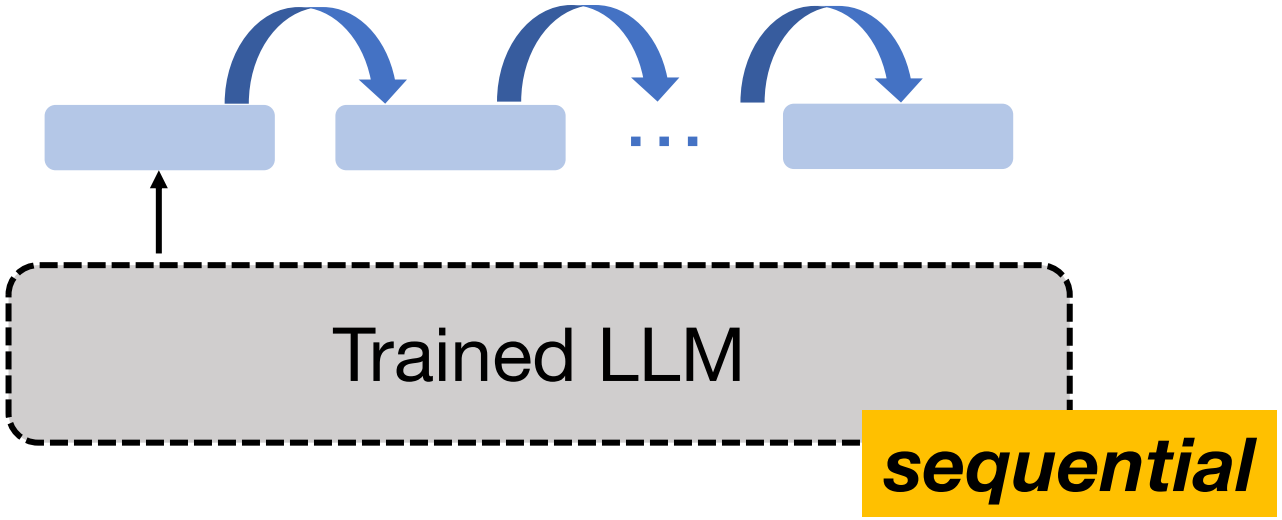
Charlie Snell[♦, 1], Jaehoon Lee[2], Kelvin Xu[♣, 2] and Aviral Kumar[♣, 2]
[♣]Equal advising, [1]UC Berkeley, [2]Google DeepMind, [♦]Work done during an internship at Google DeepMind

Key idea: Finetune LLMs to enable this behavior!

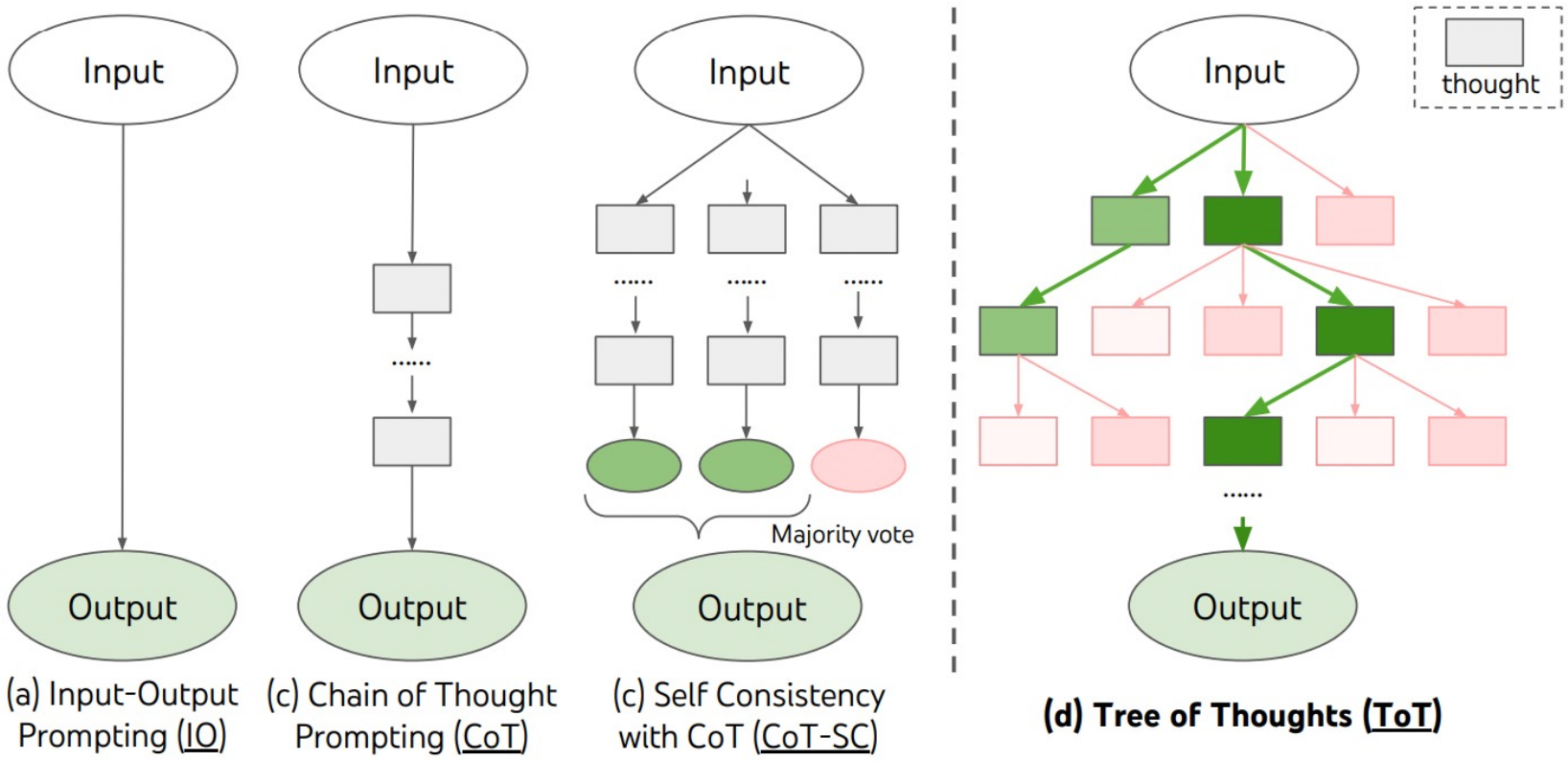


Sequential self-correction

(Qu et al. 2024, Kumar et al. 2024)



A Family of Test-Time Scaling Algorithms



This training has been done via:

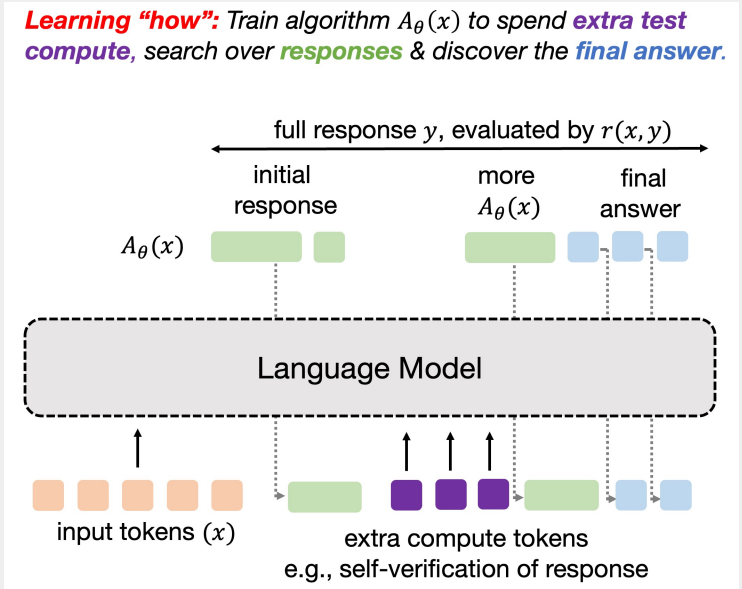
- RL or SFT
- Dense vs sparse supervision
- Different "types" of training data
- ….

Training for test-time scaling: Train LLMs to implement these procedures

Yao et al. **Tree of Thoughts.** NeurIPS 2023.

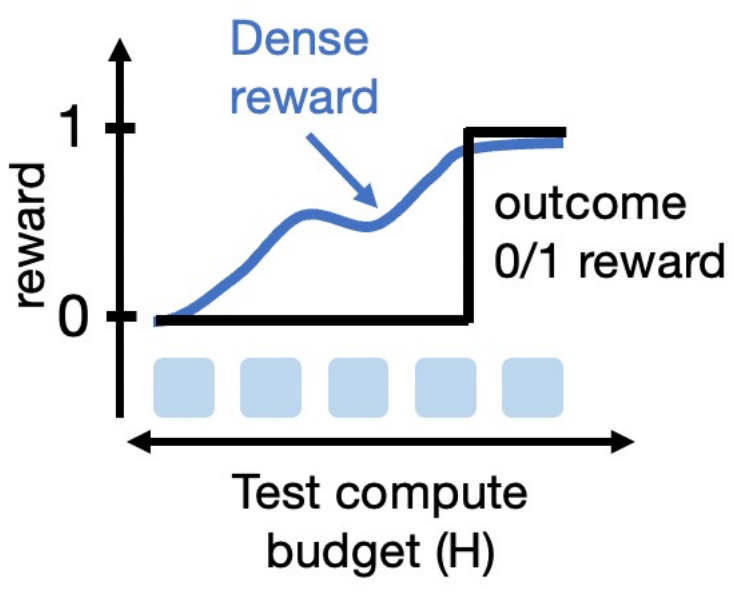
This Talk: Scaling Test-Time Compute

Desiderata and Formulation



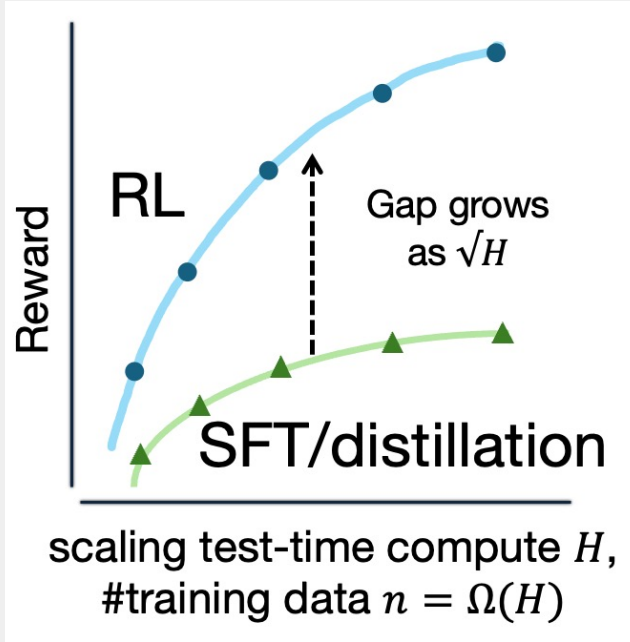
Summary: Pose it as an adaptation problem

Ingredient 1: Dense rewards



Summary: More than outcome reward needed

Ingredient 2: RL >> SFT



Summary: Use reward signals for training

Yuxiao Qu
(PhD @ CMU)

Amrith Setlur
(PhD @ CMU)

Matthew Yang
(MS @ CMU)

Nived Rajaraman
(PhD @ Berkeley)

Formulating a Learning Problem

Main paper and blog post covered:

- Scaling Test-Time Compute without Verification or RL is Suboptimal
  Setlur, Rajaraman, Levine, Kumar. arXiv 2025

- Optimizing Test-Time Compute requires Solving a Meta-RL Problem
  Setlur, Qu, Yang, Zhang, Smith, Kumar. CMU MLD Blog, 2025.

Desiderata: What We Want at Test Time

Easy input problems

- Don't spend too many tokens in arriving at the solution!
  (Also referred to as "don't overthink")

Hard input problems

- Make "progress" on hard problems
  - *Explore multiple strategies*
  - *Exploit some promising ones, retry in different ways if needed*

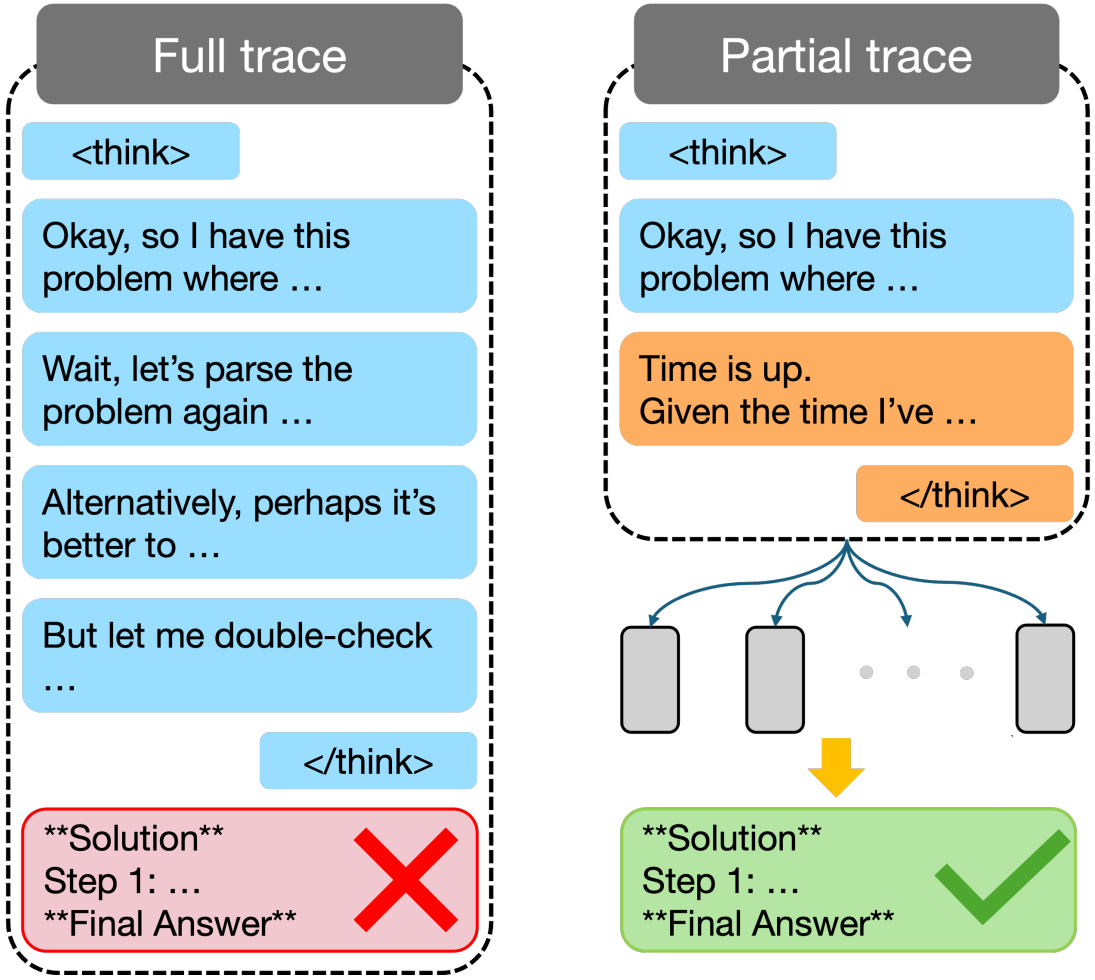
Do Current Models Enjoy these Desiderata?

Short answer: *not really!*

- **Easy problems:** Make sure to be efficient
- **Hard problems:** Make sure to make constant *progress*

Experiment setup

Chop the thinking block in DeepSeek-R1 and ask it to produce best answer



Qu*, Yang* et al. **Optimizing Test-Time Compute via Meta Reinforcement Finetuning.** arXiv 2025.

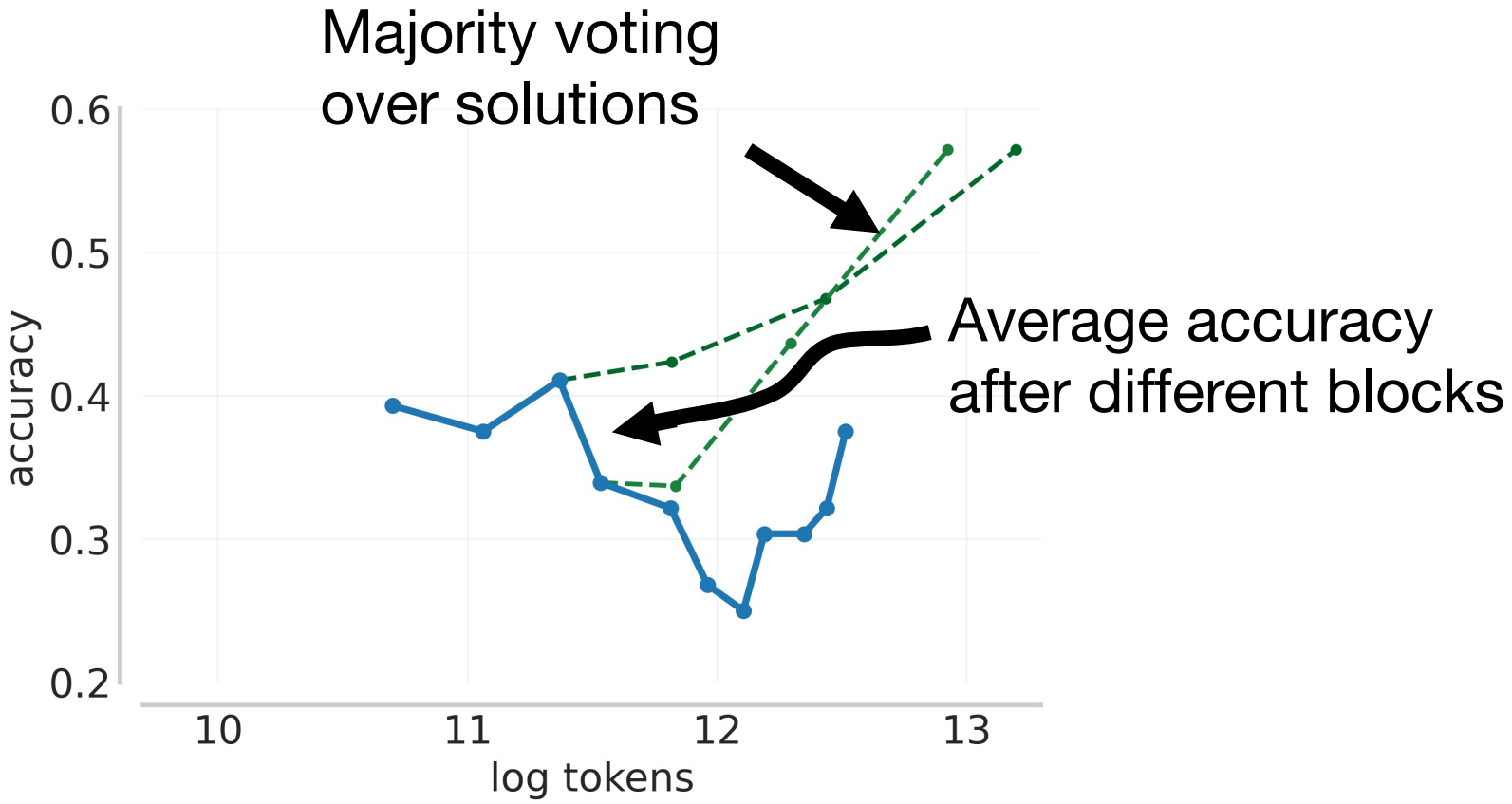
Do Current Models Enjoy these Desiderata?

Short answer: *not really!*

- **Easy problems:** Make sure to be *token*-efficient!
- **Hard problems:** Make sure to make constant *progress*

Experiment setup

Chop the thinking block in DeepSeek-R1 and ask it to produce best answer

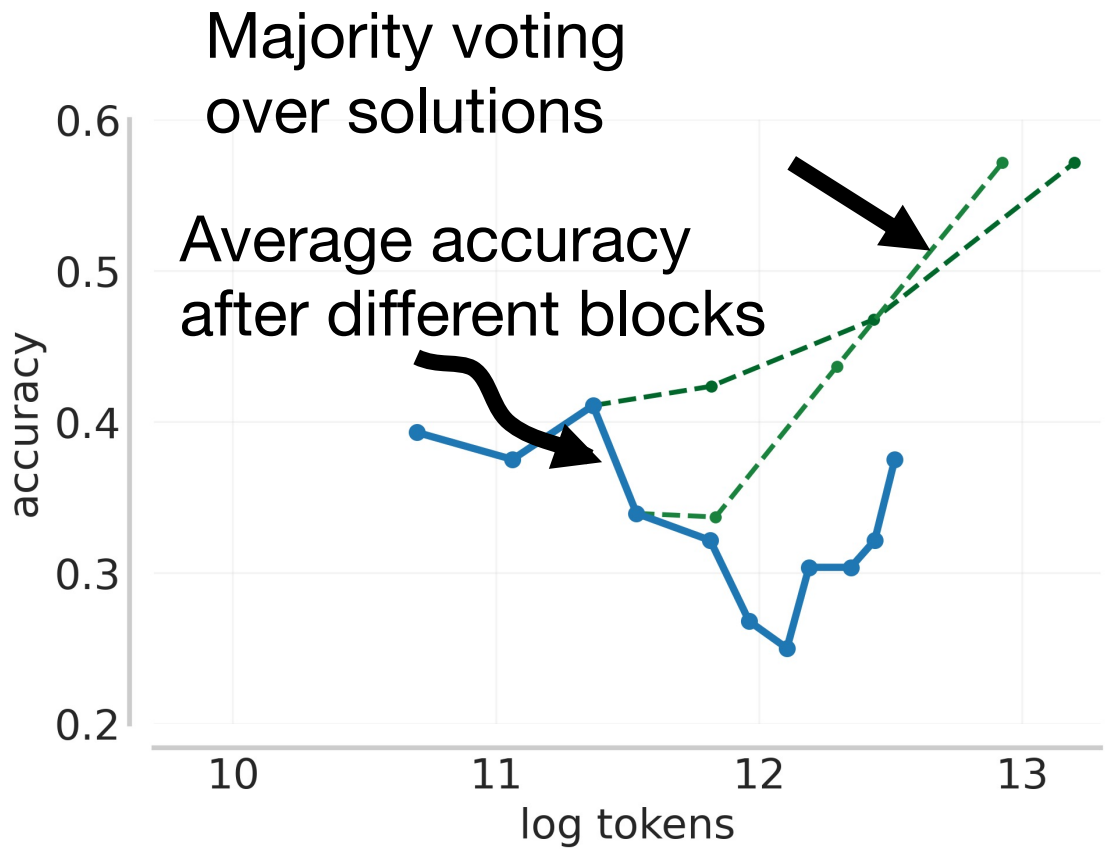


Qu*, Yang* et al. **Optimizing Test-Time Compute via Meta Reinforcement Finetuning.** arXiv 2025.

Do Current Models Enjoy these Desiderata?

Short answer: *not really!*

- **Easy problems:** Make sure to be *token*-efficient!
- **Hard problems:** Make sure to make constant *progress*



Takeaway: Can make progress by implementing the "algorithm" of running a simple majority vote, *but it does not.*

Qu*, Yang* et al. **Optimizing Test-Time Compute via Meta Reinforcement Finetuning.** arXiv 2025.

Formulation: How to Satisfy These Desiderata

Let's start from the final goal

$$\max_{\pi} \mathbb{E}_{\mathbf{x}\sim\mathcal{D}_{\text{test}}}\left[\mathbb{E}_{\mathbf{z}\sim\pi(\cdot|\mathbf{x})}\left[r(\mathbf{x},\mathbf{z})\right]\right]$$

on test problems

response sampled from model (longer than typical solution)

Total compute constraint per problem

Qu*, Yang* et al. **Optimizing Test-Time Compute via Meta Reinforcement Finetuning.** arXiv 2025.

Formulation: How to Satisfy These Desiderata

$$\max_{\pi} \mathbb{E}_{\mathbf{x}\sim\mathcal{D}_{\text{test}}}\left[\mathbb{E}_{\mathbf{z}\sim\pi(\cdot|\mathbf{x})}\left[r(\mathbf{x},\mathbf{z})\right]\right] \text{ s.t. } \forall\mathbf{x}, \mathbb{E}_{\pi(\cdot|\mathbf{x})}|\mathbf{z}| \leq C_0$$

But this compute budget is fixed!

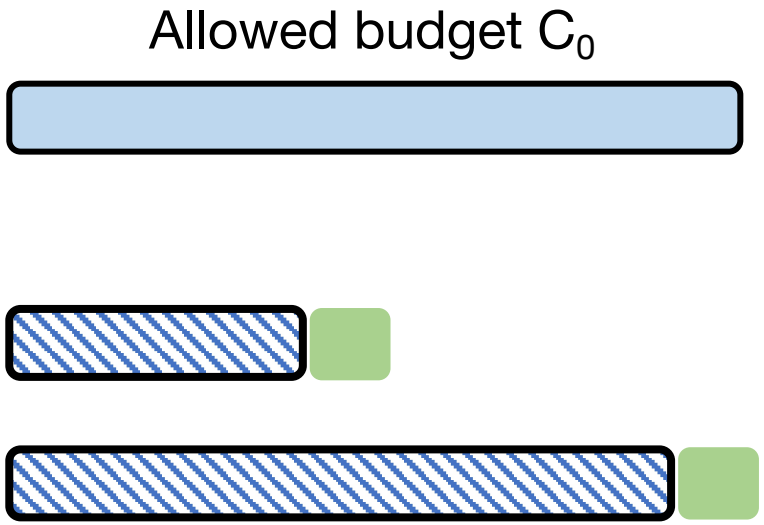
$$\max_{\pi} \mathbb{E}_{\mathbf{x}\sim\mathcal{D}_{\text{train}}}\left[\mathbb{E}_{\mathbf{z}\sim\pi(\cdot|\mathbf{x})}\left[r(\mathbf{x},\mathbf{z})\right]\right] \text{ s.t. } \ldots$$

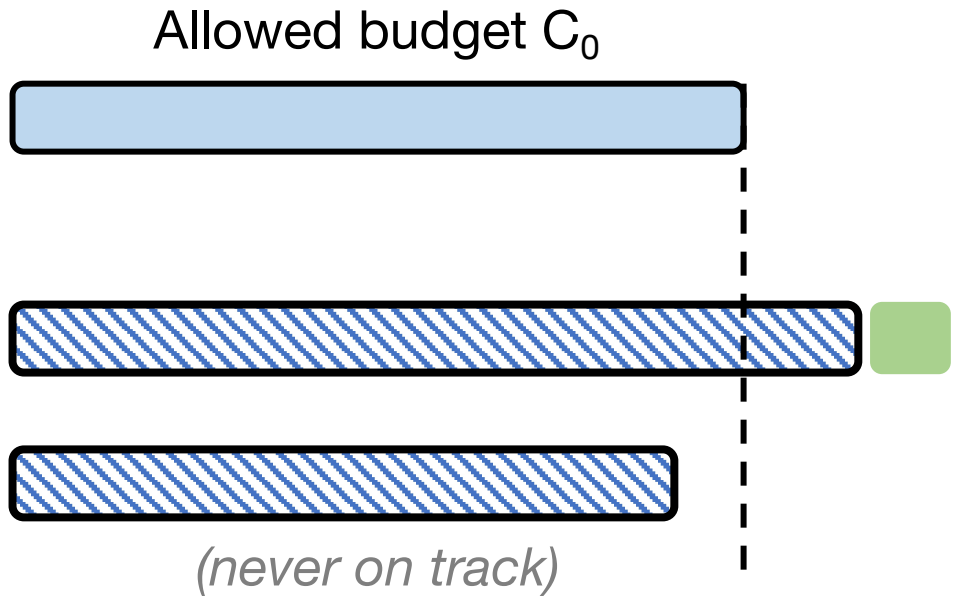
Can optimize this via:

- RL (like DeepSeek-R1): outcome-reward RL
- SFT / STaR: collect data, filter by correctness, maximize likelihood

Qu*, Yang* et al. **Optimizing Test-Time Compute via Meta Reinforcement Finetuning.** arXiv 2025.

Why is Outcome Reward + Fixed Budget Bad?





Easy problems: Both get rewarded the same way

Hard problems: Neither trace gets rewarded

Qu*, Yang* et al. **Optimizing Test-Time Compute via Meta Reinforcement Finetuning.** arXiv 2025.

Formulation: "Budget-Agnostic" LLMs

Key idea: *Incentivize the LLM to make progress regardless of the compute budget*

$$\max_{\pi} \ \mathbb{E}_{\mathbf{x}\sim\mathcal{D}_{\text{train}}}\left[\mathbb{E}_{\mathbf{z}\sim\pi(\cdot|\mathbf{x})}\left[r(\mathbf{x},\mathbf{z})\right]\right] \ \text{s.t.} \ \ldots$$

$$r(\mathbf{x},\mathbf{z}) + \sum_{i=1}^{H} r_{\text{prg}}(\mathbf{x},\mathbf{z}_{0:i})$$

Some segment of the entire trace

Some dense reward to incentivize progress

Qu*, Yang* et al. **Optimizing Test-Time Compute via Meta Reinforcement Finetuning.** arXiv 2025.

Open Questions: Budget-Agnostic Training

$$r(\mathbf{x}, \mathbf{z}) + \sum_{i=1}^{H} r_{\text{prg}}(\mathbf{x}, \mathbf{z}_{0:i})$$

Concrete design

What's a good design for r_{prg} to use?

How to optimize

Should I use RL training or just SFT / STaR?

Ingredient 1: Dense rewards

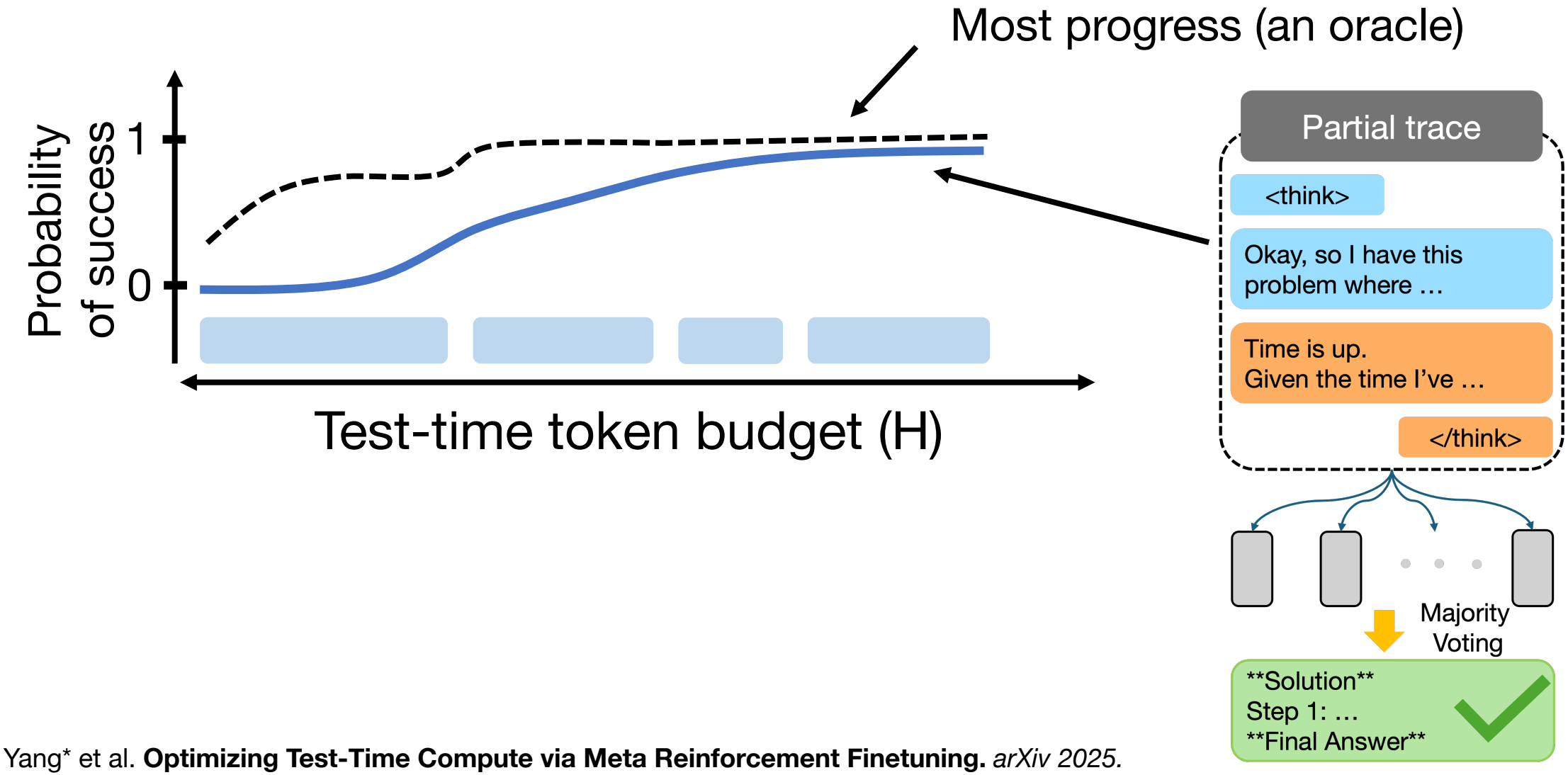
Designing Dense Rewards

Main paper covered:

- Optimizing Test-Time Compute via Meta Reinforcement Finetuning.
  Qu, Yang*, Setlur, Tunstall, Beeching, Salakhutdinov, Kumar. arXiv 2025*

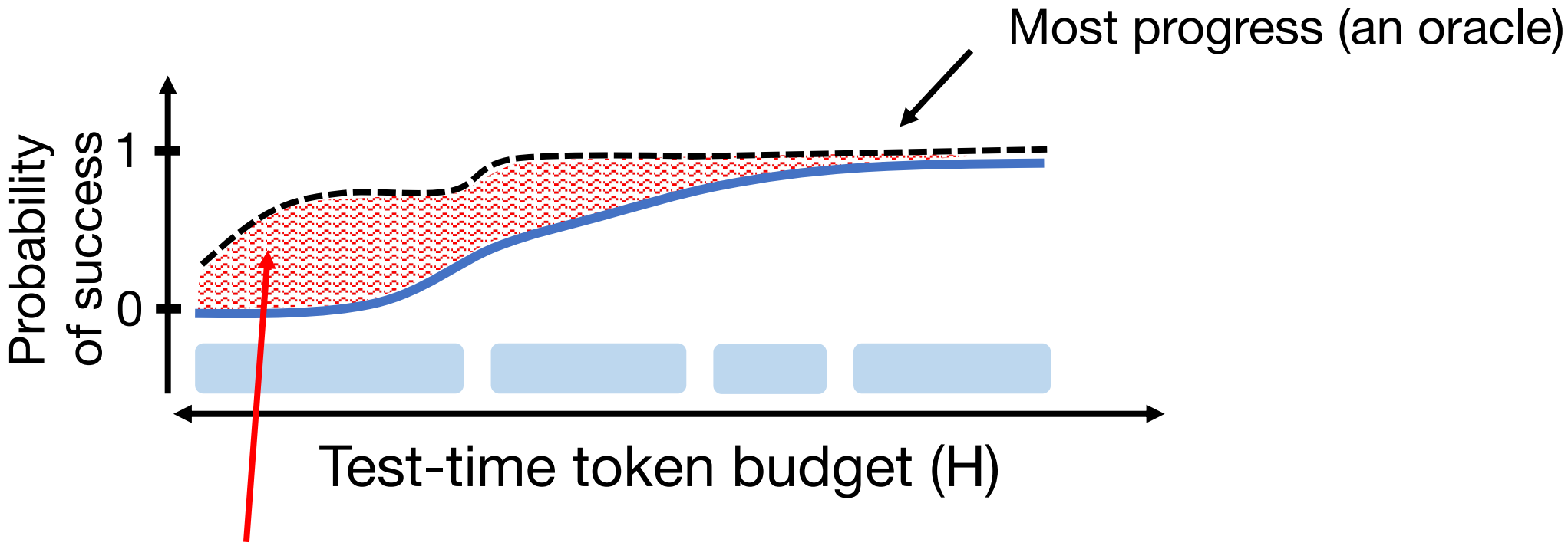
- **Also see:** Rewarding Progress: Scaling Automated Process Verifiers for LLM Reasoning
  Setlur, Nagpal, Fisch, Geng, Eisenstein, R. Agarwal, A. Agarwal, Berant, Kumar. ICLR 2025 (Spotlight).

Idea: What is Good Progress on New Problems?



Qu*, Yang* et al. **Optimizing Test-Time Compute via Meta Reinforcement Finetuning.** *arXiv 2025.*

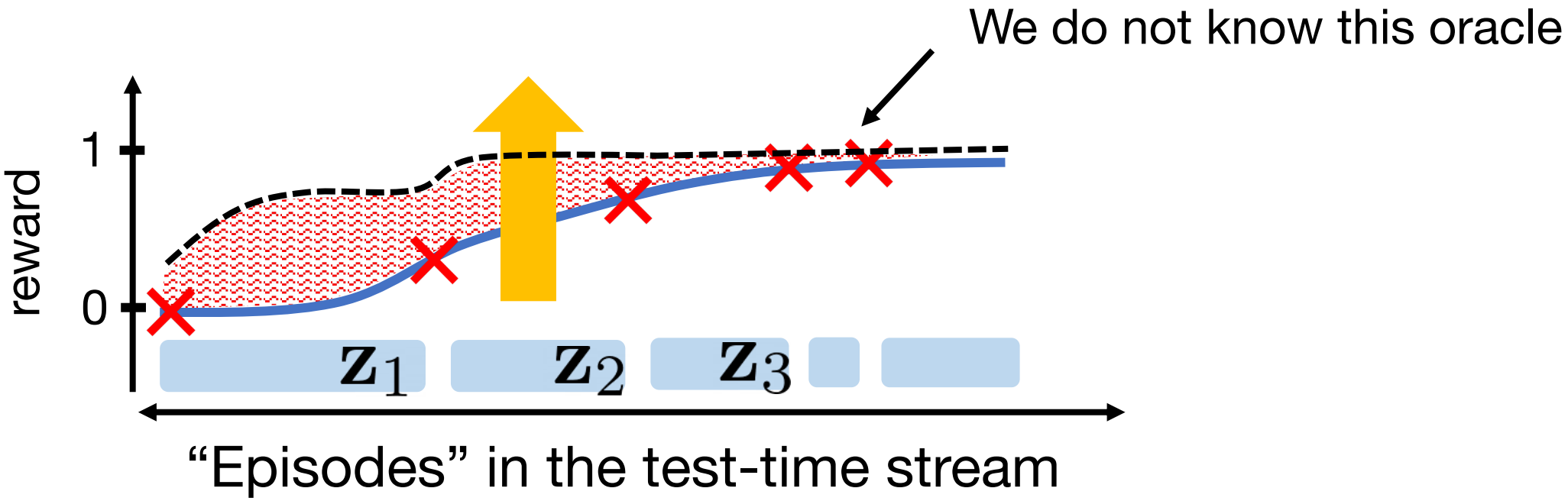
Idea: What is Good Progress on New Problems?



We want this area to be as low as possible!
(i.e., a notion of cumulative regret)

Qu*, Yang* et al. **Optimizing Test-Time Compute via Meta Reinforcement Finetuning.** *arXiv 2025.*

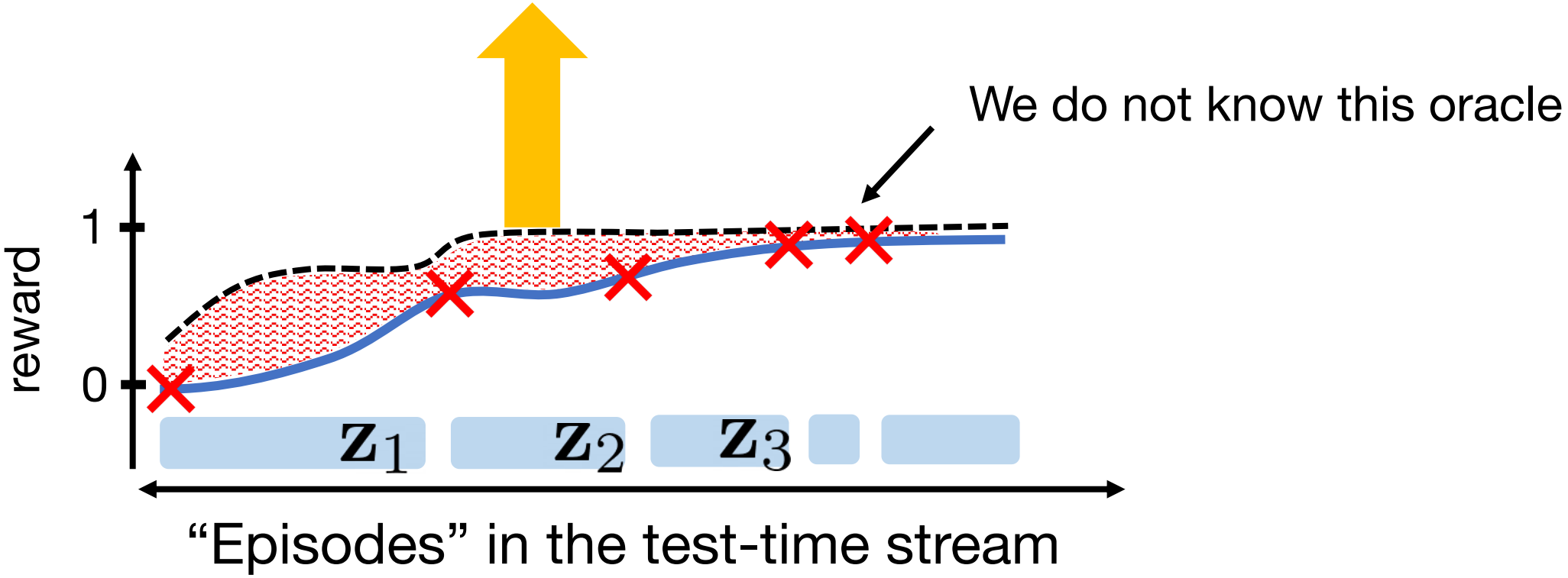
Inducing a Good Curve That Makes Progress



Key idea: Can still push up the performance after every episode!

Qu*, Yang* et al. **Optimizing Test-Time Compute via Meta Reinforcement Finetuning.** *arXiv 2025.*

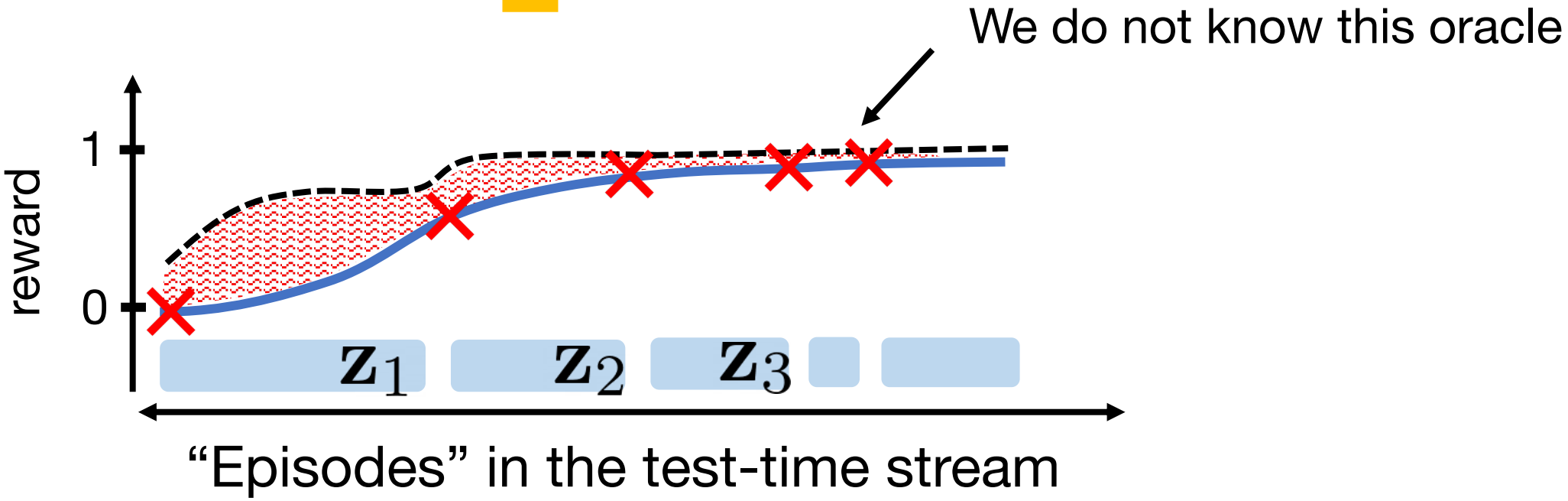
Inducing a Good Curve That Makes Progress



Key idea: Can still push up the performance after every episode!

Qu*, Yang* et al. **Optimizing Test-Time Compute via Meta Reinforcement Finetuning.** *arXiv 2025.*

Inducing a Good Curve That Makes Progress



Key idea: Can still push up the performance after every episode!

Qu*, Yang* et al. **Optimizing Test-Time Compute via Meta Reinforcement Finetuning.** *arXiv 2025.*

Concrete Idea: Progress Reward Design

$$\max_{\pi} \ \mathbb{E}_{\mathbf{x}\sim\mathcal{D}_{\text{train}}}\left[\mathbb{E}_{\mathbf{z}\sim\pi(\cdot|\mathbf{x})}\left[r(\mathbf{x},\mathbf{z})\right]\right] \ \text{s.t. } \ldots$$

Convert to policy gradient

over episodes

$$\mathbb{E}_{\mathbf{x}\sim\mathcal{D}_{\text{train}},\mathbf{z}\sim\pi(\cdot|\mathbf{x})}\left[\sum_j \nabla_\pi \log \pi(\mathbf{z}_j|\mathbf{x},\mathbf{z}_{0:j-1}) \cdot r(\mathbf{x},\mathbf{z})\right]$$

Add progress reward to the gradient

$$\mathbb{E}_{\mathbf{x}\sim\mathcal{D}_{\text{train}},\mathbf{z}\sim\pi(\cdot|\mathbf{x})}\left[\sum_j \nabla_\pi \log \pi(\mathbf{z}_j|\mathbf{x},\mathbf{z}_{0:j-1}) \cdot \left(r(\mathbf{x},\mathbf{z}) + \alpha \cdot r_{\text{prg}}(\mathbf{x},\mathbf{z}_{0:j})\right)\right]$$

Progress reward

Qu*, Yang* et al. **Optimizing Test-Time Compute via Meta Reinforcement Finetuning.** *arXiv 2025.*

Concrete Idea: Progress Reward Design

$$\mathbb{E}_{\mathbf{x}\sim\mathcal{D}_{\text{train}},\mathbf{z}\sim\pi(\cdot|\mathbf{x})}\left[\sum_j \nabla_\pi \log \pi(\mathbf{z}_j|\mathbf{x},\mathbf{z}_{0:j-1})\cdot(r(\mathbf{x},\mathbf{z})+\alpha\cdot r_{\text{prg}}(\mathbf{x},\mathbf{z}_{0:j}))\right]$$

So far, we were using accuracy of the model
that aims to write out the solution given the episodes so far

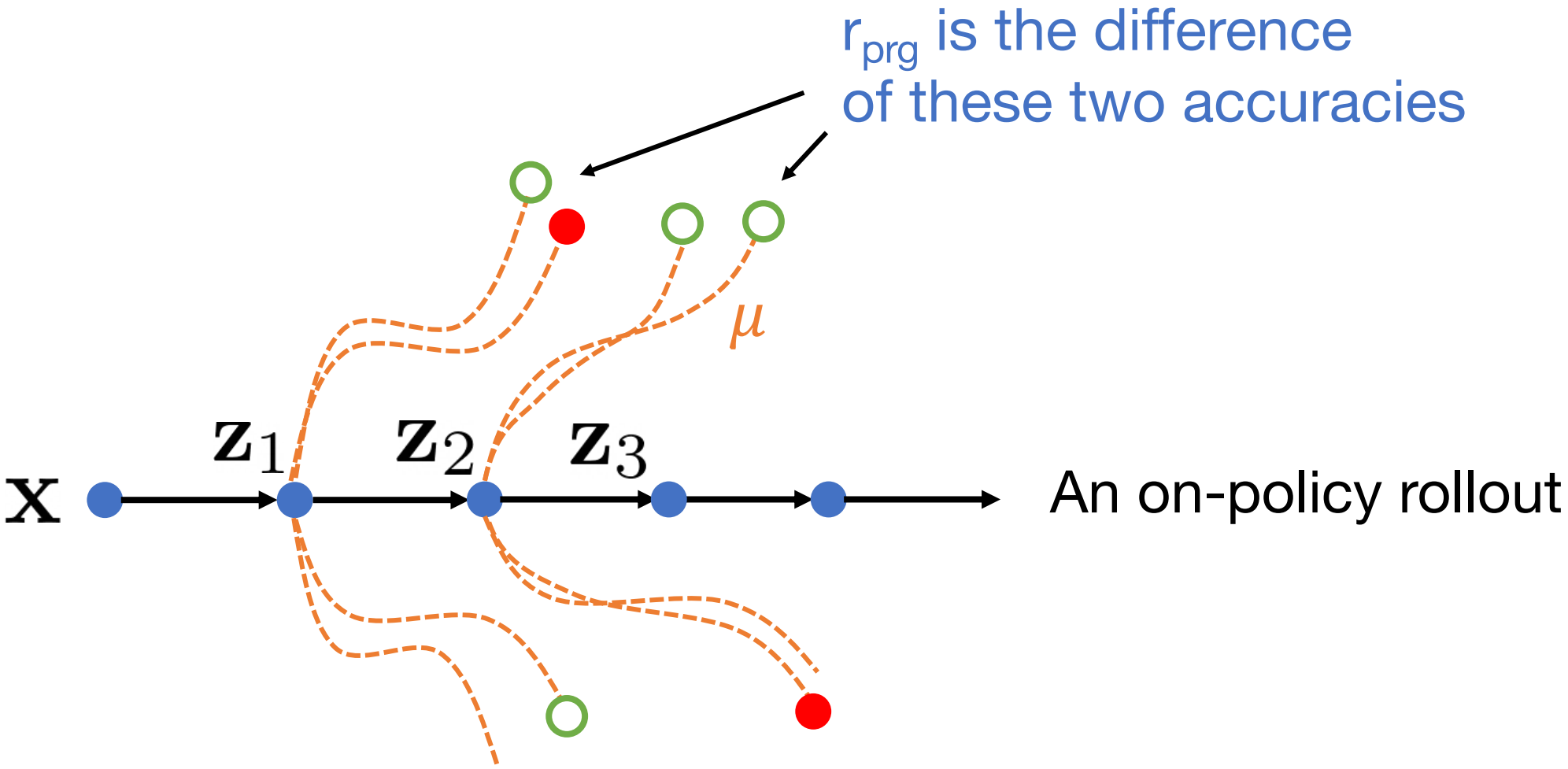
Progress reward

A "prover" policy that guesses the best answer

$$r_{\text{prg}}(\mathbf{x},\mathbf{z}_{0:j}) = J_r(\mu(\cdot|\mathbf{x},\mathbf{z}_{0:j})) - J_r(\mu(\cdot|\mathbf{x},\mathbf{z}_{0:j-1}))$$

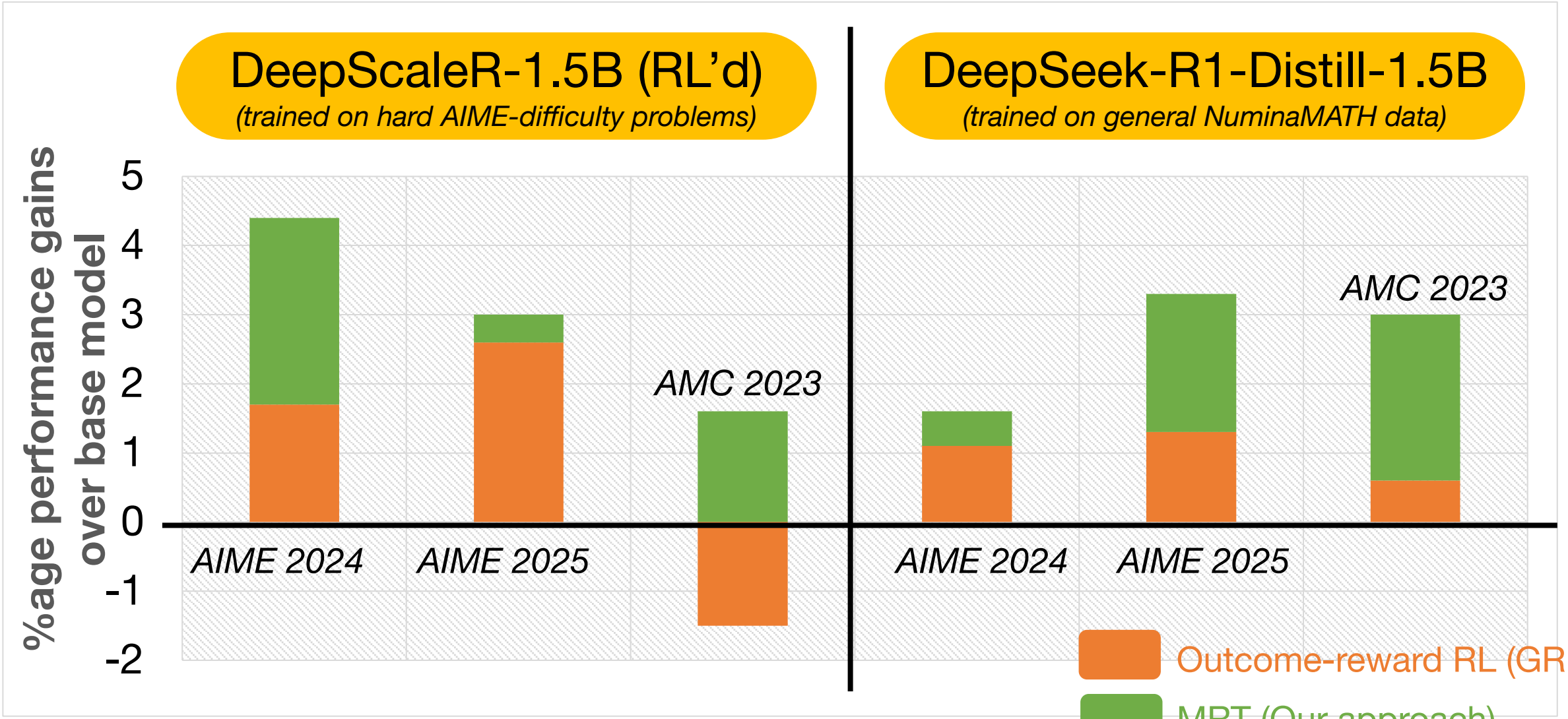
Qu*, Yang* et al. **Optimizing Test-Time Compute via Meta Reinforcement Finetuning.** *arXiv 2025.*

Concrete Idea: Progress Reward Design



Qu*, Yang* et al. **Optimizing Test-Time Compute via Meta Reinforcement Finetuning.** *arXiv 2025.*

(Subset of) Results: Our Approach (MRT)



Qu*, Yang* et al. **Optimizing Test-Time Compute via Meta Reinforcement Finetuning.** *arXiv 2025.*
Luo*, Tan* et al. **DeepScaleR: Surpassing O1-Preview with a 1.5B Model by Scaling RL.** 2025.

Open Questions: Dense Rewards

See: RL on Incorrect Synthetic Data Scales the Efficiency of LLM Math Reasoning 8x. NeurIPS 2024.

Computational cost

- Estimating dense rewards requires rollouts, which are costly.
  - *Can we get more juice out of the same total FLOPs??*

The choice of the prover policy

- The policy μ determines the progress reward.
  - *How should you choose this policy??*

See: Rewarding Progress: Scaling Automated Process Verifiers. ICLR 2025.

Other ways of implementing the same principle

- Length curriculum and iterative training could be one other way
  - *Many open-source implementations kinda do this!*

Qu*, Yang* et al. **Optimizing Test-Time Compute via Meta Reinforcement Finetuning.** *arXiv 2025.*

Ingredient 2:
RL >> SFT

Comparing RL to SFT for test-time scaling

Main paper covered:

- Scaling Test-Time Compute without Verification or RL is Suboptimal
  Setlur, Rajaraman, Levine, Kumar. arXiv 2025

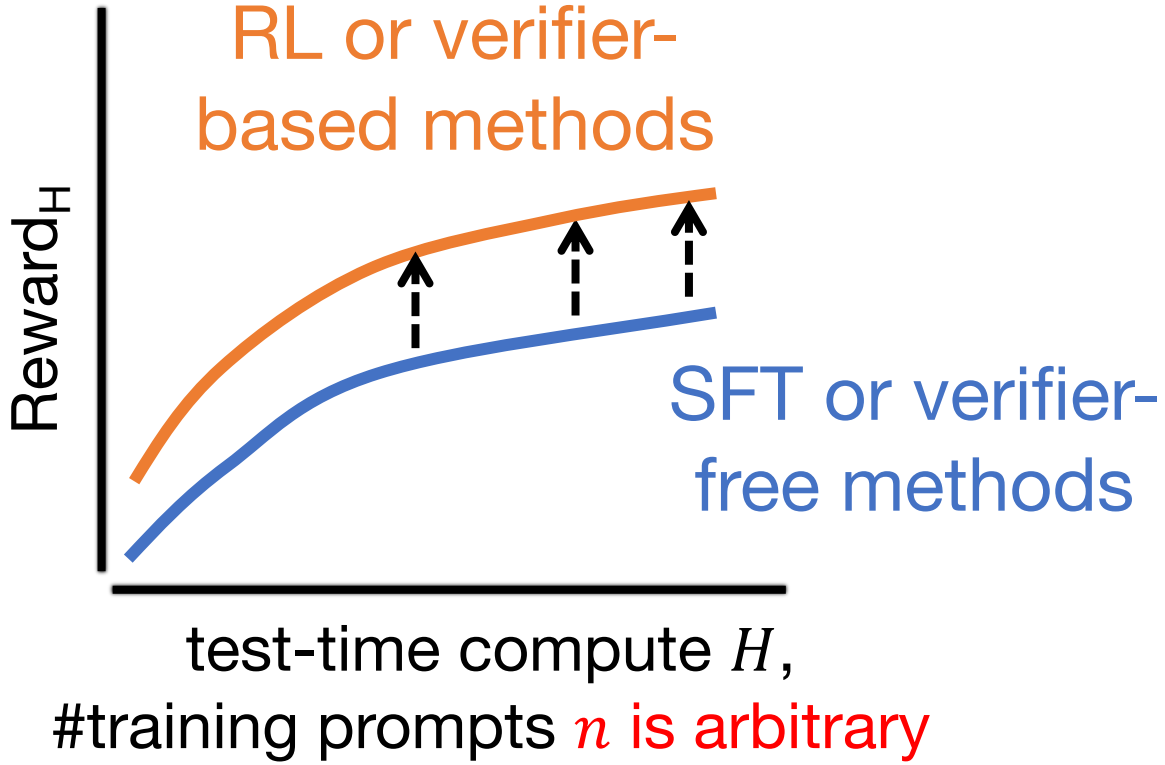
Goal: Scaling Test-Time Compute Effectively

Good performance as we allow the model to use more compute

For this analysis, we use a simple notion of binary 0/1 (step) progress

Theory: Using rewards via RL > cloning via SFT

When pre-trained LLM satisfies ***certain criteria***



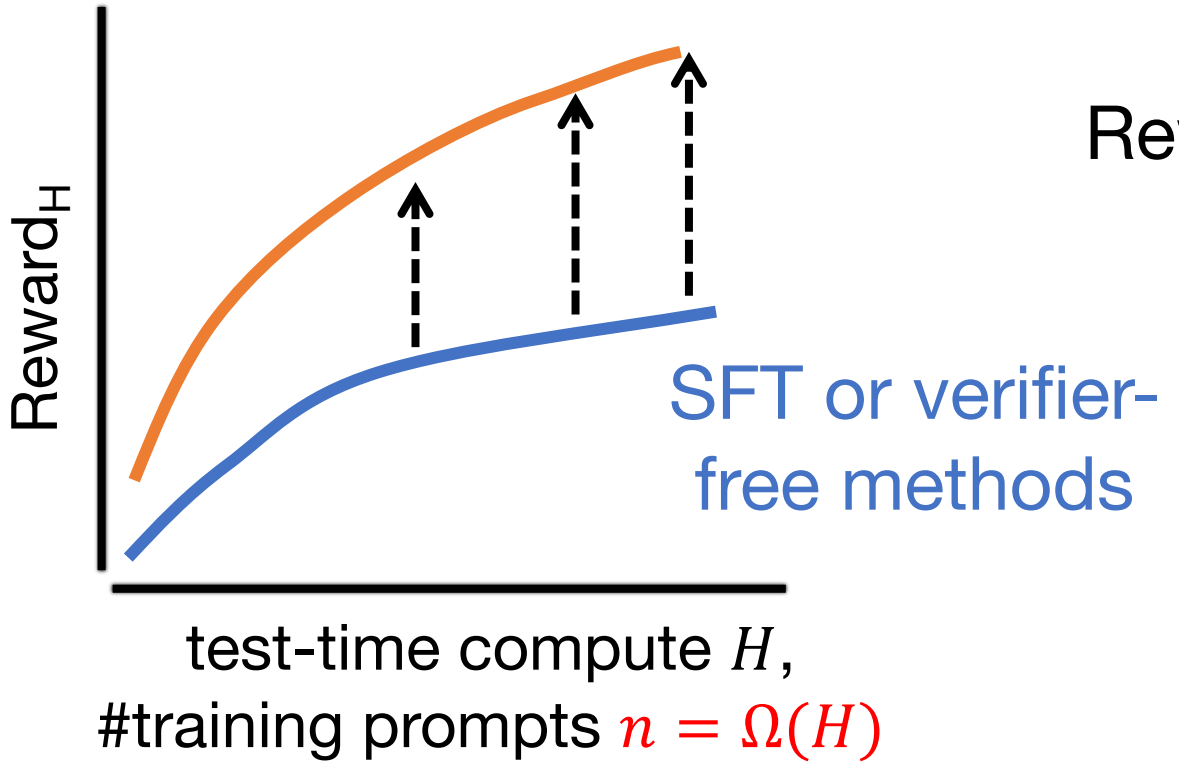
$$\text{Reward}_H(RL) - \text{Reward}_H(SFT) > 0$$

Result 1: Gap between using rewards vs. not is positive at big H values

Setlur et al. **Scaling Test-Time Compute Without Verification or RL is Suboptimal.** arXiv 2025.

Theory: Using rewards via RL > cloning via SFT

When pre-trained LLM satisfies ***certain criteria***



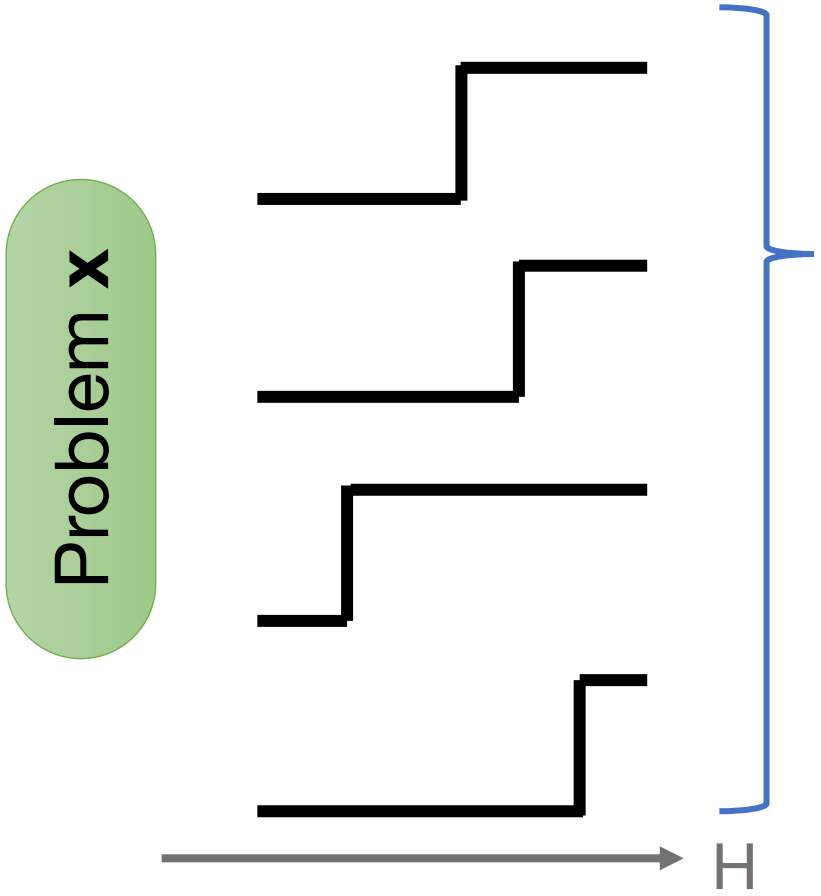
$$\text{Reward}_H(RL) - \text{Reward}_H(SFT) = \Omega(\sqrt{H})$$

Result 2: If we scale test compute and number of prompts, RL will do lot better than SFT

Setlur et al. **Scaling Test-Time Compute Without Verification or RL is Suboptimal.** arXiv 2025.

Criterion 1: Base LLM is Heterogeneous

Intuition: Diversity of traces under dense reward, *per problem*



All of these solve the problem…

..but make progress at different rates

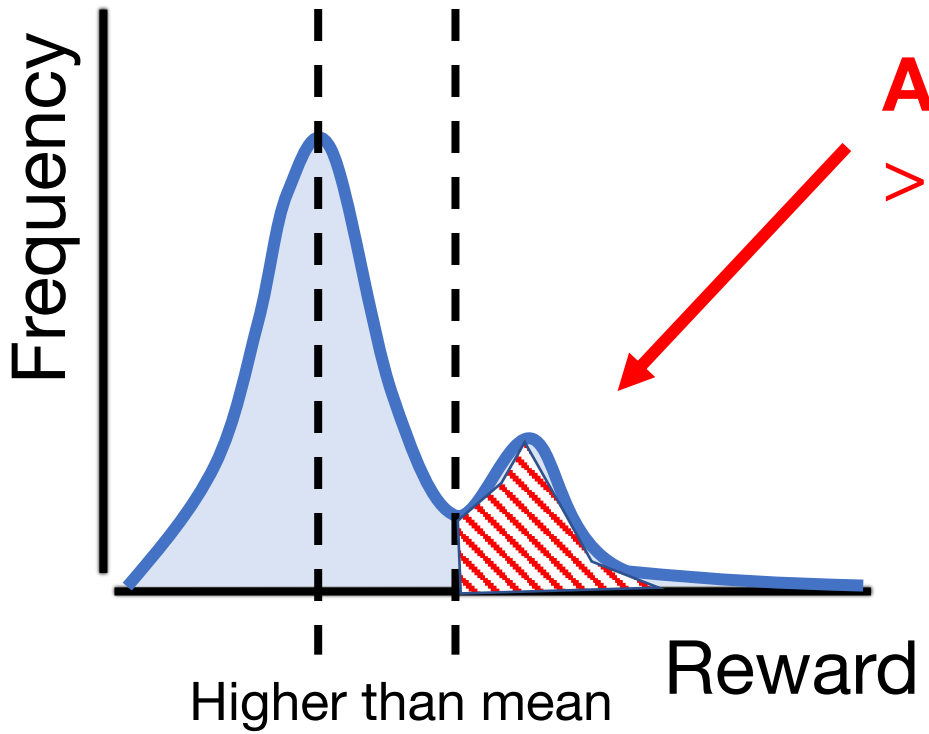
This hurts SFT
(or verifier-free) methods.

Setlur et al. **Scaling Test-Time Compute Without Verification or RL is Suboptimal.** arXiv 2025.

Criterion 2: Base LLM is "Anticoncentrated"

Intuition: High enough (constant) coverage of better traces

Sample a bunch of traces and plot the ***reward histogram***:



Anticoncentration:
>= constant probability in "better" regions.

This enables RL to explore to find a good policy with training.

Setlur et al. **Scaling Test-Time Compute Without Verification or RL is Suboptimal.** arXiv 2025.

Theory: Putting It Together

[Informal] Separation b/w verifier-based & verifier-free

For the notion of step-level progress reward, when training with *compute-budget H* and *data size n, we see:*

1. **Lower bound for VF (SFT):** $\text{SubOpt}_H(\text{VF}) = \Omega\left(\frac{\sigma}{\sqrt{n}}\right), \sigma \approx \Omega(H)$

Variance of dense rewards on a given problem

Setlur et al. **Scaling Test-Time Compute Without Verification or RL is Suboptimal.** arXiv 2025.

Theory: Putting It Together

[Informal] Separation b/w verifier-based & verifier-free

For the notion of step-level progress reward, when training with *compute-budget H* and *data size n, we see:*

1. **Lower bound for VF (SFT):** $\text{SubOpt}_H(\text{VF}) = \Omega\left(\frac{\sigma}{\sqrt{n}}\right), \sigma \approx \Omega(H)$

2. **Upper bound for VB (RL):** $\text{SubOpt}_H(\text{VB}) = 0\left(\frac{1}{c_{base}} \cdot \frac{H}{n}\right)$

Anti-concentration probability mass

Error in learning the reward signal

Setlur et al. **Scaling Test-Time Compute Without Verification or RL is Suboptimal.** arXiv 2025.

Theory: Putting It Together

[Informal] Separation b/w verifier-based & verifier-free

For the notion of step-level progress reward, when training with *compute-budget H* and *data size n, we see:*

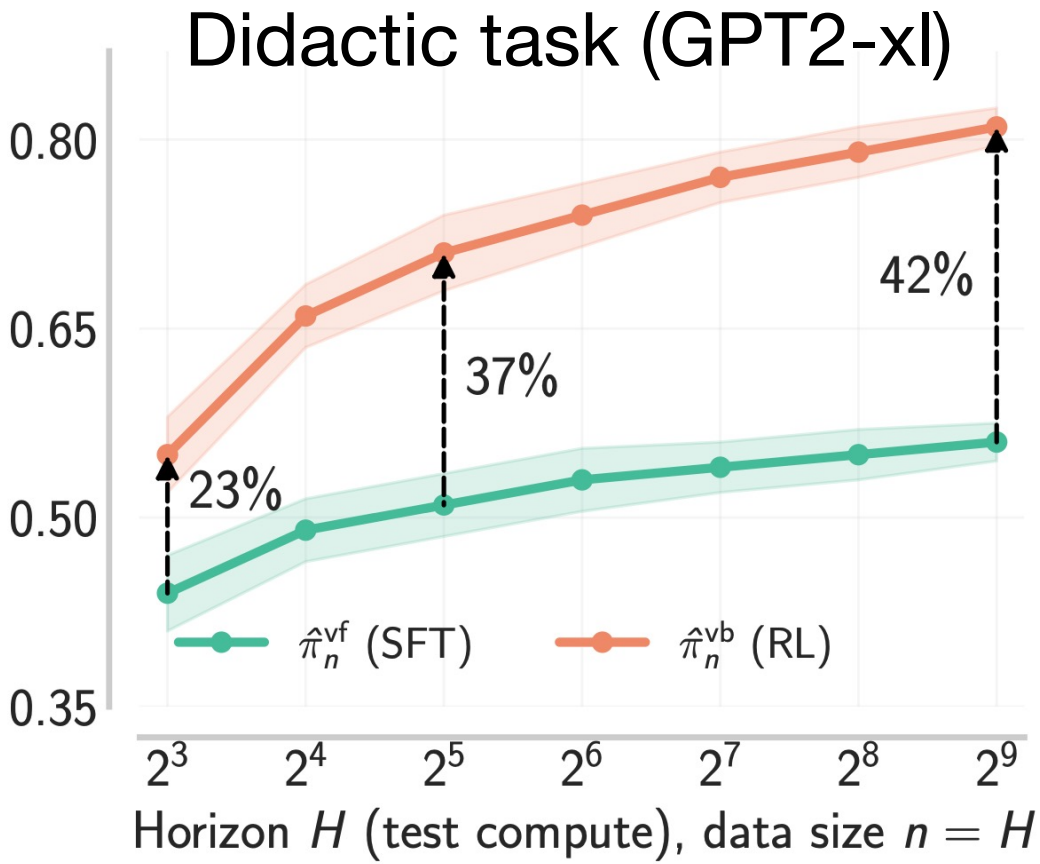
1. **Lower bound for VF (SFT):** $\text{SubOpt}_H(\text{VF}) = \Omega\left(\frac{\sigma}{\sqrt{n}}\right), \sigma \approx \Omega(H)$

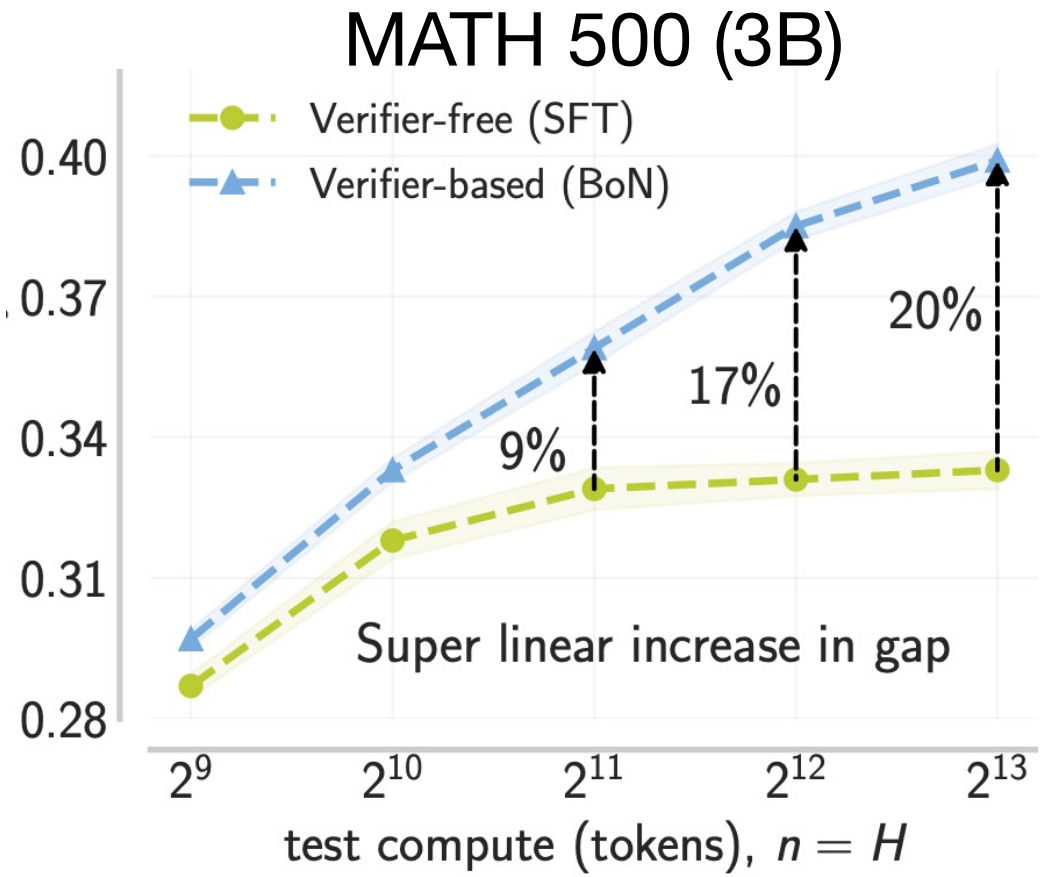
2. **Upper bound for VB (RL):** $\text{SubOpt}_H(\text{VB}) = 0\left(\frac{1}{c_{base}} \cdot \frac{H}{n}\right)$

3. **Gap b/w RL and SFT** scales as $\Omega\left(\frac{H}{\sqrt{n}}\right)$. Set $n = 0(H)$.

Setlur et al. **Scaling Test-Time Compute Without Verification or RL is Suboptimal.** arXiv 2025.

Results: Scaling Data and Compute Both

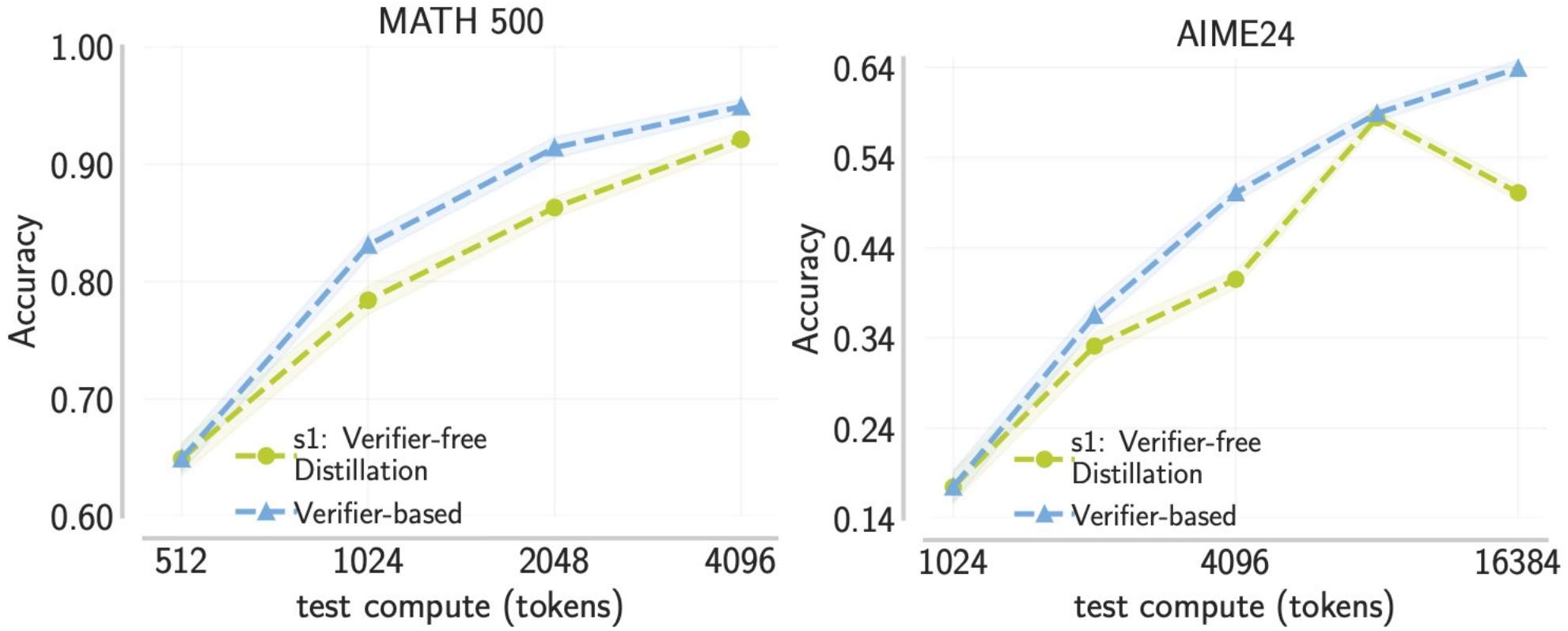




Finding: Gap between verifier-based and verifier-free methods increases

Setlur et al. **Scaling Test-Time Compute Without Verification or RL is Suboptimal.** arXiv 2025.

Results: Scaling Compute, Not Data (w/ s1-32B)



Finding: Gap between using reward verification and SFT is positive

Setlur et al. **Scaling Test-Time Compute Without Verification or RL is Suboptimal.** arXiv 2025.

RL Training

Takeaways

- **Key idea:** RL (using reward signal) scales test-time compute better than SFT, with a notion of dense rewards.

- **Main findings:**
  - When data is scaled linearly in H, performance gap grows
  - When data is kept the same, RL performs better still.
  - Base LLMs tend to be heterogeneous & anti-concentrated.

Thank You! Questions?

Scaling Test-Time Compute Without Verification or RL is Suboptimal

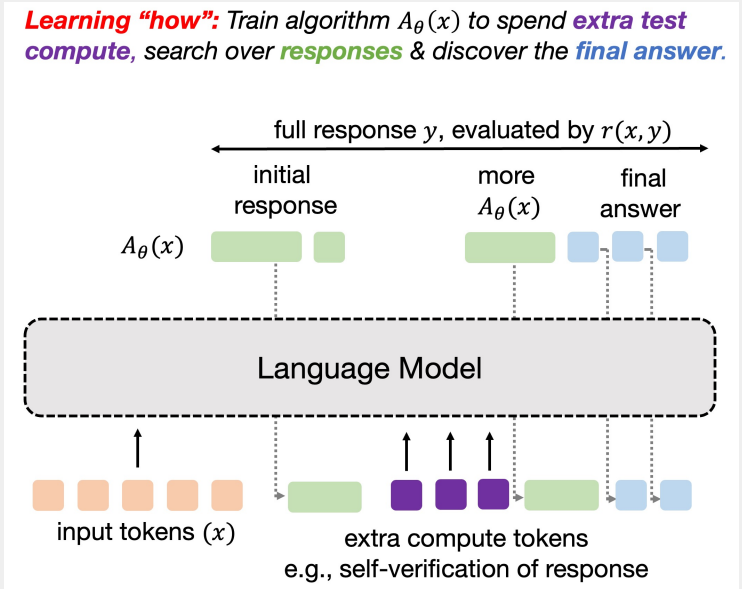
Amrith Setlur[1], **Nived Rajaraman**[2], **Sergey Levine**[2] **and Aviral Kumar**[1]
[1]Carnegie Mellon University, [2]UC Berkeley

Optimizing Test-Time Compute via Meta Reinforcement Fine-Tuning

Yuxiao Qu[*1], **Matthew Y. R. Yang**[*1], **Amrith Setlur**[1], **Lewis Tunstall**[2], **Edward Emanuel Beeching**[2], **Ruslan Salakhutdinov**[1] **and Aviral Kumar**[1]
[1]Carnegie Mellon University, [2]Hugging Face, [*]Equal contribution

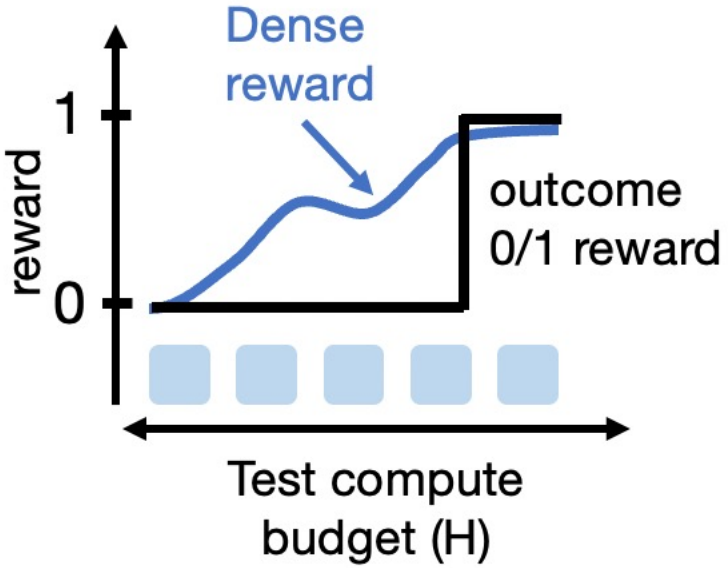
Summary: Scaling Test-Time Compute

Budget-agnostic formulation



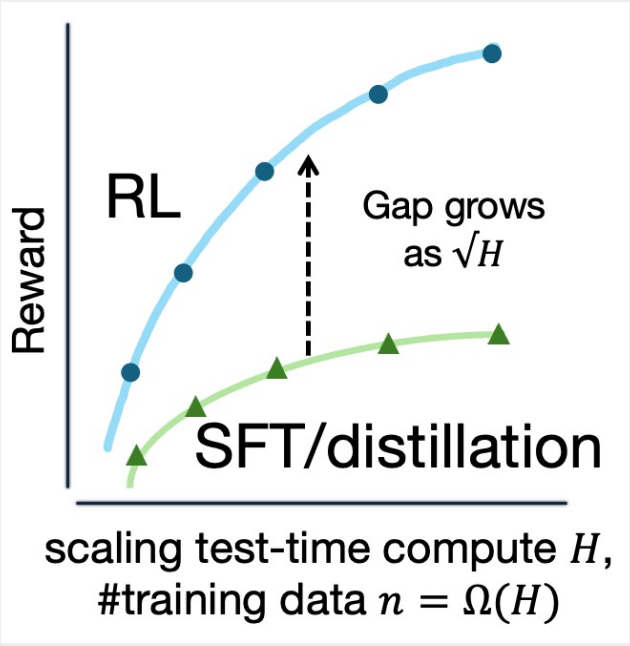
Summary: Budget-agnostic training

MRT incorporates dense rewards



Summary: Use dense rewards w/ progress

RL scales much better that SFT



Summary: Use reward signals with RL / verifiers