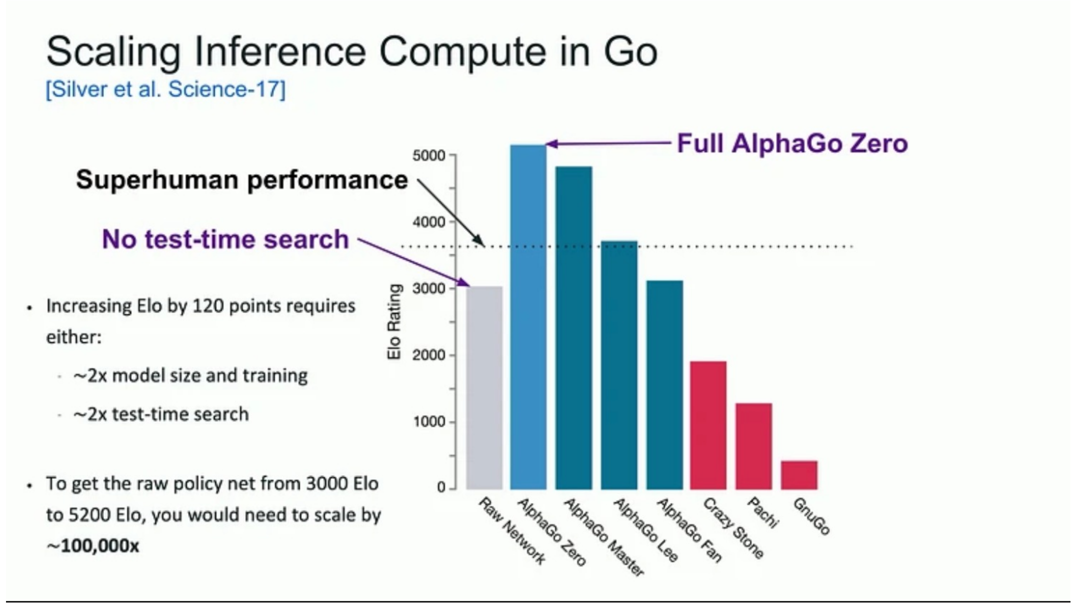
Dynamic decomposition Improves Inference Scaling (DISC)

Domain agnostic, step-by-step decomposition of LLM reasoning

Why is search important?

- Influential in achieving superhuman performance for AlphaGo
- Instead of following just one chain of thought, the model can try several reasoning paths to find better answers



Scaling Inference Compute in Go
[Silver et al. Science-17]

- Increasing Elo by 120 points requires either:
  - ~2x model size and training
  - ~2x test-time search
- To get the raw policy net from 3000 Elo to 5200 Elo, you would need to scale by **~100,000x**

"The two methods that seem to scale arbitrarily […] are search and learning"

-- Richard Sutton

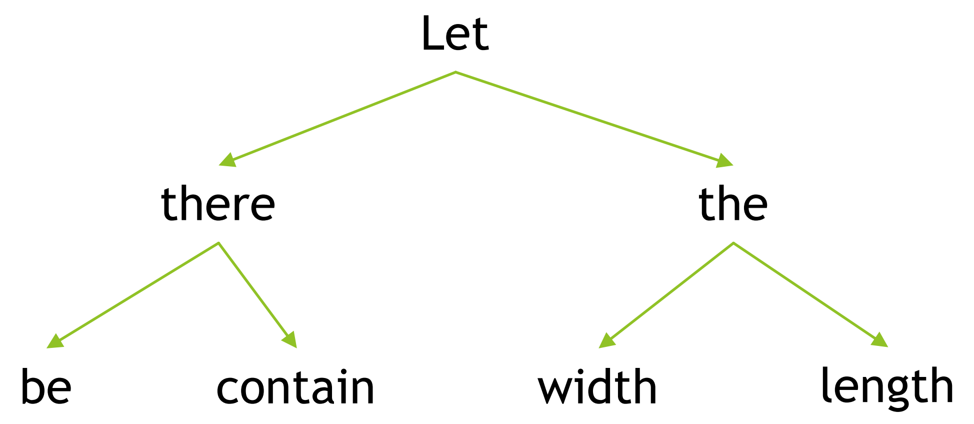
Key research question

How do you conduct search over reasoning traces for LLMs?

Example tree search

Search requires partitioning a solution into steps

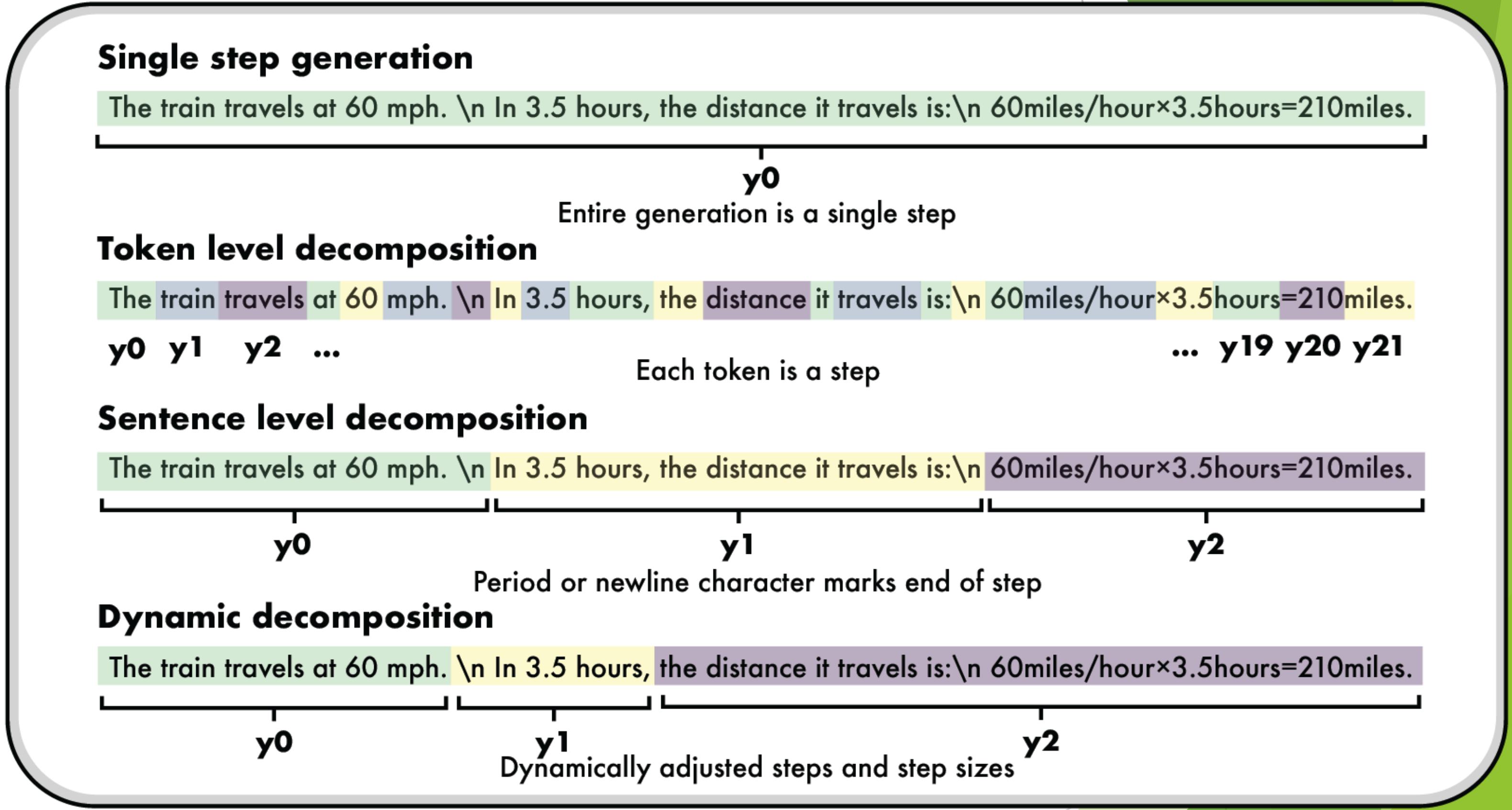
Calculate the maximum area of a rectangle with perimeter 24



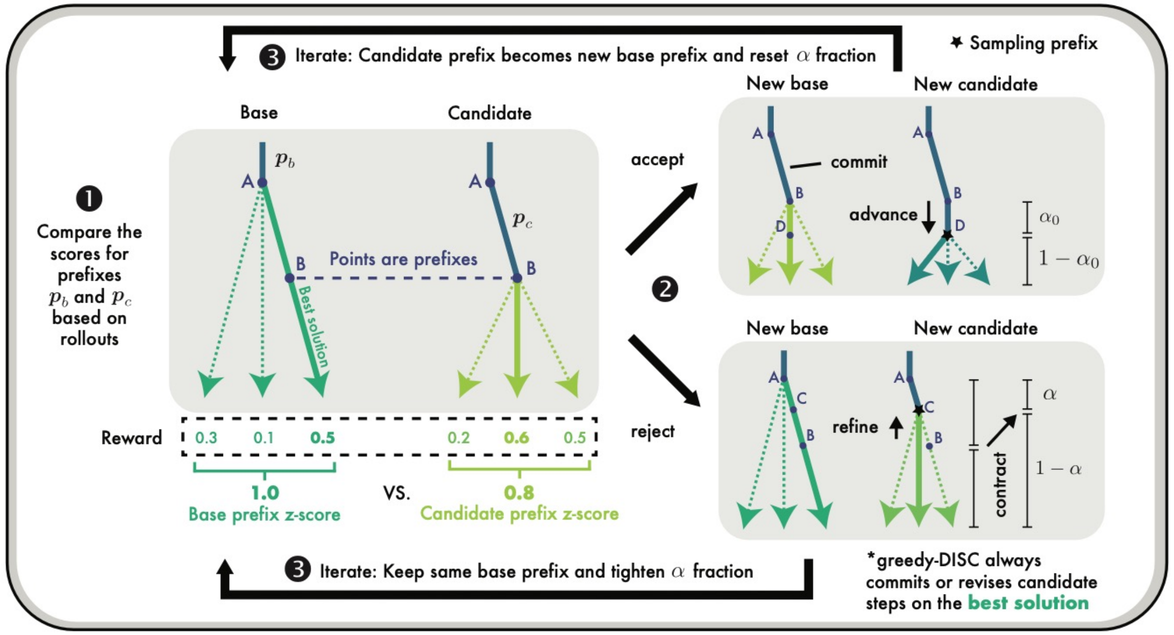
The search tree quickly becomes intractable as both the branching factor (~100K tokens) and depth (>1K tokens) explode in the language space

Solution

- Search across sequences of tokens (i.e., reasoning “steps”)
- Break down into steps that tailored for the LLM reasoning
- Spend more compute searching over more “difficult” steps


Single step generation
The train travels at 60 mph. \n In 3.5 hours, the distance it travels is:\n 60miles/hour×3.5hours=210miles.
y0
Entire generation is a single step
Token level decomposition
The train travels at 60 mph. \n In 3.5 hours, the distance it travels is:\n 60miles/hour×3.5hours=210miles.
y0 y1 y2 ...
Each token is a step
... y19 y20 y21
Sentence level decomposition
The train travels at 60 mph. \n In 3.5 hours, the distance it travels is:\n 60miles/hour×3.5hours=210miles.
y0
y1
y2
Period or newline character marks end of step
Dynamic decomposition
The train travels at 60 mph. \n In 3.5 hours, the distance it travels is:\n 60miles/hour×3.5hours=210miles.
y0
y1
y2
Dynamically adjusted steps and step sizes

DISC Algorithm

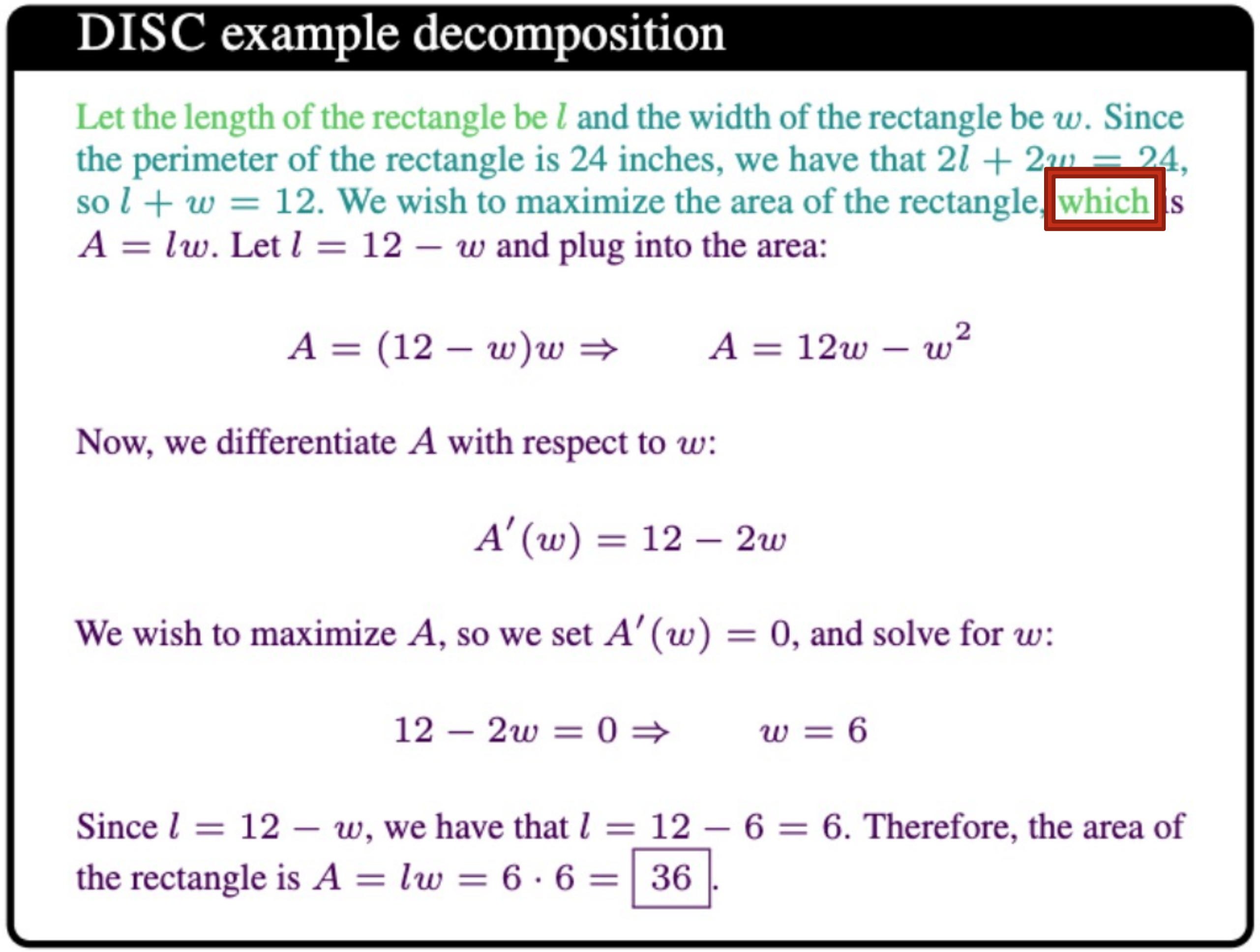


③ Iterate: Candidate prefix becomes new base prefix and reset α fraction
★ Sampling prefix
Base
Candidate
❶ Compare the scores for prefixes p_b and p_c based on rollouts
p_b
p_c
A
B
Points are prefixes
Best solution
Reward
0.3 0.1 0.5
0.2 0.6 0.5
1.0 Base prefix z-score
vs.
0.8 Candidate prefix z-score
accept
reject
②
New base
New candidate
commit
advance
D
α_0
$1-\alpha_0$
C
refine
contract
α
$1-\alpha$
③ Iterate: Keep same base prefix and tighten α fraction
*greedy-DISC always commits or revises candidate steps on the best solution

Figure 2: **DISC with Greedy Search.** One iteration of Alg. 1. We start with a base prefix A and a candidate prefix B. We compare the sample statistics of each by evaluating a scoring function (e.g., z-score). If the candidate prefix B demonstrates a higher likelihood of reward improvement compared to continuing from base prefix A, we accept B, commit it as the new base, and extend the candidate to a further step (e.g., BD) on the best sampled solution. If B is not better, we reject it and propose a shorter candidate (e.g., AC), contracting the step size. This process repeats until a new candidate is accepted or all options are exhausted. The algorithm thus adaptively advances or contracts the step size and search horizon based on the relative quality of completions from each prefix.

- Dynamically proposes step sizes
- Uses feedback from the LLM generations to guide step decomposition

Example DISC decomposition



DISC example decomposition

Let the length of the rectangle be l and the width of the rectangle be w. Since the perimeter of the rectangle is 24 inches, we have that $2l + 2w = 24$, so $l + w = 12$. We wish to maximize the area of the rectangle, which is $A = lw$. Let $l = 12 - w$ and plug into the area:

$$A = (12 - w)w \Rightarrow \qquad A = 12w - w^2$$

Now, we differentiate A with respect to w:

$$A'(w) = 12 - 2w$$

We wish to maximize A, so we set $A'(w) = 0$, and solve for w:

$$12 - 2w = 0 \Rightarrow \qquad w = 6$$

Since $l = 12 - w$, we have that $l = 12 - 6 = 6$. Therefore, the area of the rectangle is $A = lw = 6 \cdot 6 = \boxed{36}$.

- “Which” is a keyword for LLM reasoning!

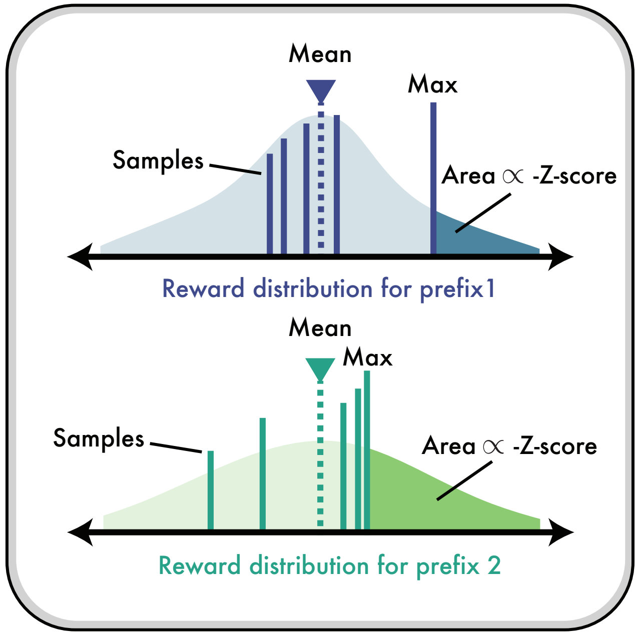

Mean
Max
Samples
Area ∝ -Z-score
Reward distribution for prefix1
Mean
Max
Samples
Area ∝ -Z-score
Reward distribution for prefix 2

Figure 4: Reward distribution of rollouts sampled from two different prefixes. The probability of sampling a higher rollout from prefix 2 is higher than that of prefix 1.

DISC seeks to maximize probability of sampling correct trajectory

Instead of maximizing average reward or minimizing cumulative regret like traditional UCB

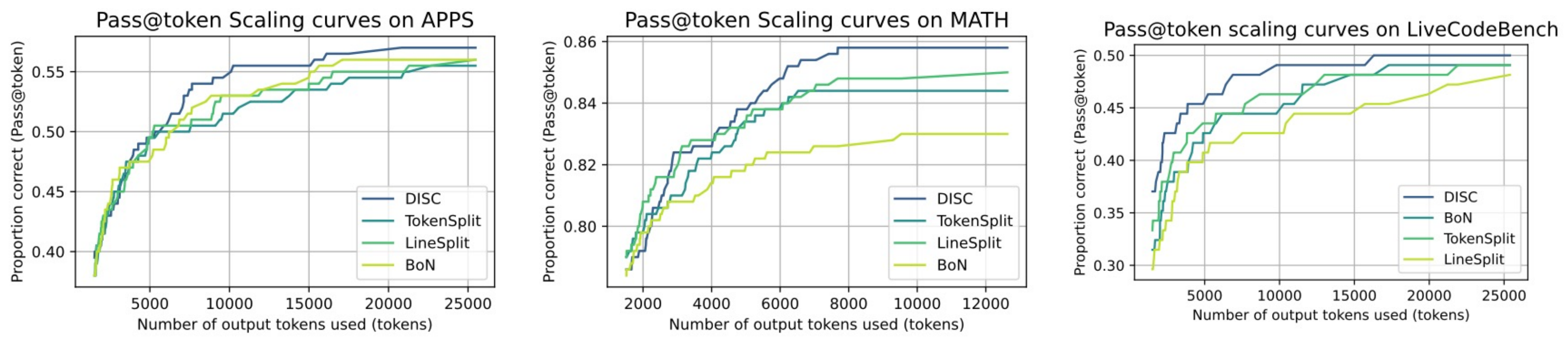


Figure 5: **Token-level comparisons across benchmarks** using gpt-4o-mini. (Left) APPS competition level (Middle) MATH500 (Right) LiveCodeBench. DISC achieves superior inference scaling over baselines on all three benchmarks.

DISC performance

5-10% accuracy gains over baseline across code and math benchmarks at almost every compute budget. Better test-time scaling

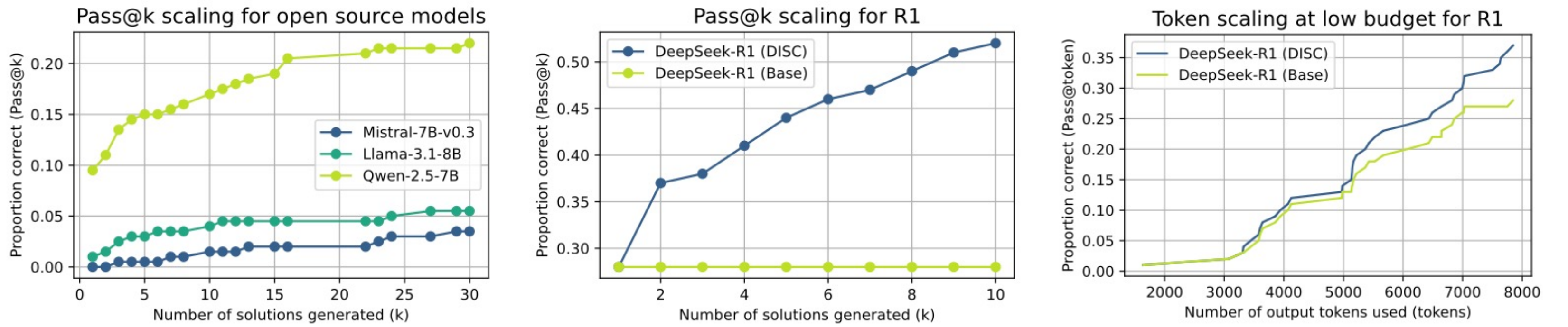


Figure 6: **Inference scaling for different models, including reasoning models, on APPS.** (Left) Pass@k for open-source models (Middle) Pass@k for R1 (Right) Pass@token for R1. DISC almost **doubles** base performance across all models.

4x gain on Llama with just 10 solutions
33% relative improvement vs DeepSeek-R1 base model at same budget!

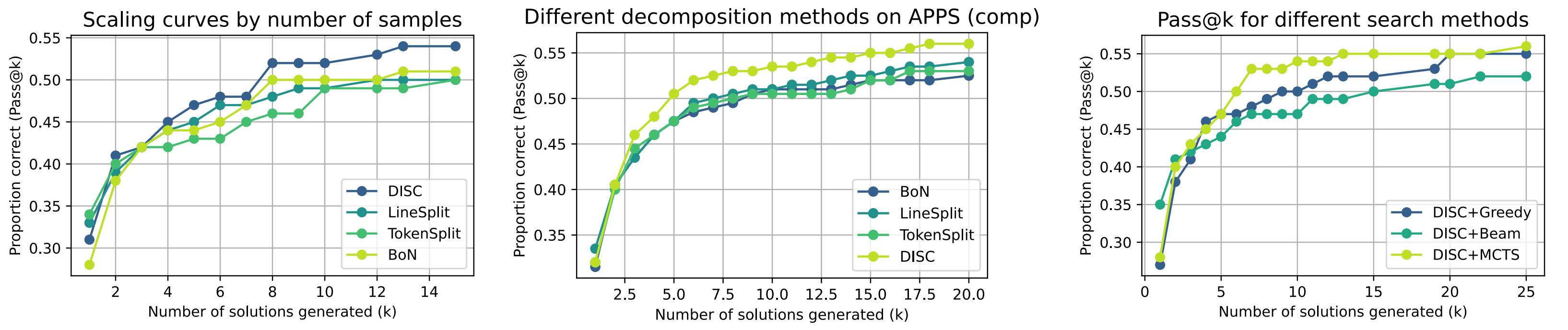


Figure 8: **Comparison of Pass@k performance on APPS using gpt-4o-mini.** (Left) self generated validation tests, (Middle) with ground truth tests, (Right) with different search methods.

DISC works also works with noisy self-generated validation tests

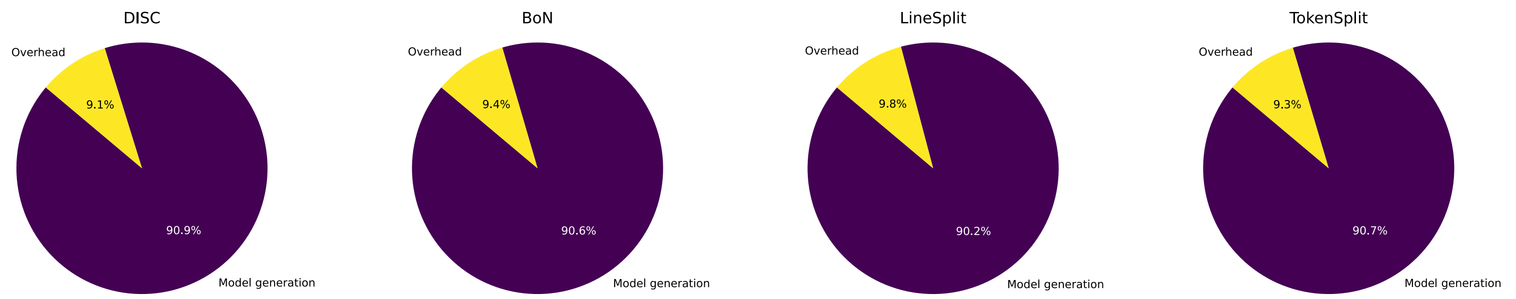


Figure 7: **Percentage of runtime spent on overhead vs LLM token generation.** DISC does not increase the runtime overhead significantly.

DISC has negligible runtime impact

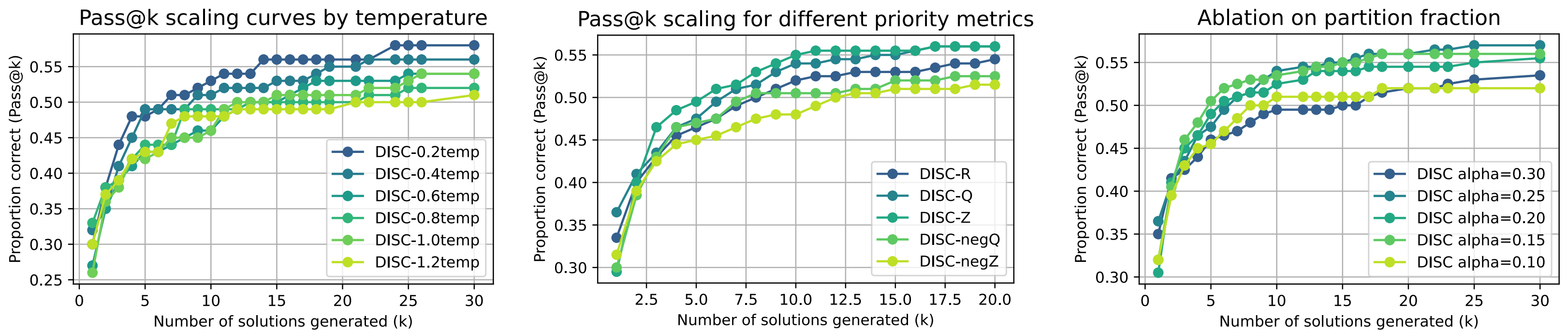
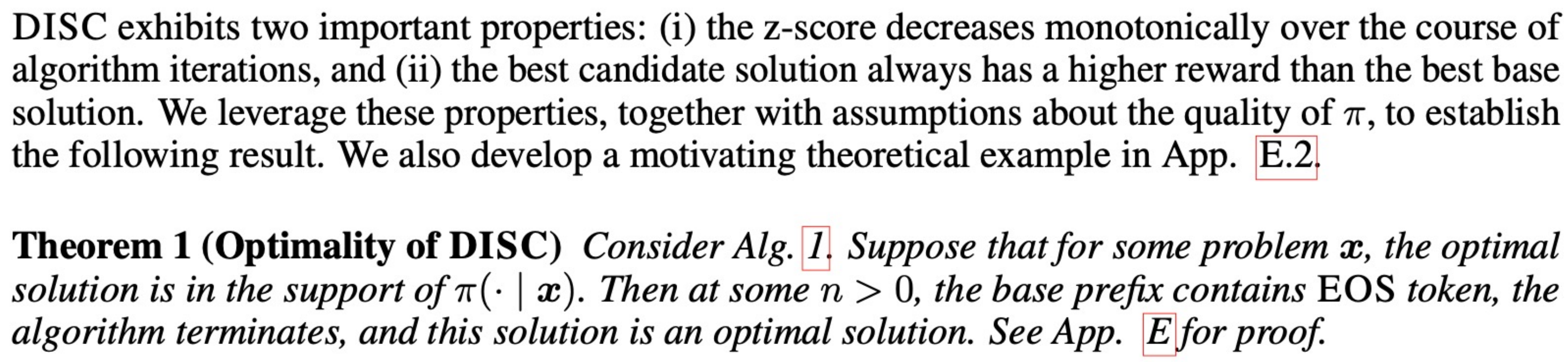


Figure 9: **Ablation study of DISC on APPS with gpt-4o-mini.** (Left) Effect of temperature: unlike BoN and other inference scaling methods, DISC achieves higher performance at lower temperatures. (Middle) effect of acceptance method (Right). Effect of partition fraction α_0: The range $0.15 \leq \alpha_0 \leq 0.25$ appears optimal.

Decreasing temperature improves DISC performance!



DISC exhibits two important properties: (i) the z-score decreases monotonically over the course of algorithm iterations, and (ii) the best candidate solution always has a higher reward than the best base solution. We leverage these properties, together with assumptions about the quality of π, to establish the following result. We also develop a motivating theoretical example in App. E.2.

Theorem 1 (Optimality of DISC) *Consider Alg. 1. Suppose that for some problem $\boldsymbol{x}$, the optimal solution is in the support of $\pi(\cdot \mid \boldsymbol{x})$. Then at some $n > 0$, the base prefix contains* EOS *token, the algorithm terminates, and this solution is an optimal solution. See App. E for proof.*

Proof of optimality of DISC

Conclusion

- Search can greatly improve LLM problem-solving capabilities
- To conduct search effectively we need to break reasoning traces into steps
- Using DISC to tailor step sizes for LLM reasoning greatly improves performance on coding and math
- DISC is provably better than common test-time methods such as BoN
- For more information, check out our website: https://disc-search.github.io/