Learning to Better Search with Language Models via Guided Reinforced Self-Training

Seungyong Moon[1], **Bumsoo Park**[2], **Hyun Oh Song**[1*]
[1]Seoul National University, [2]KRAFTON
{symoon11,hyunoh}@mllab.snu.ac.kr
bumsoo.park96@krafton.com

Abstract

While language models have shown remarkable performance across diverse tasks, they still encounter challenges in complex reasoning scenarios. Recent research suggests that language models trained on linearized search traces toward solutions, rather than solely on the final solutions, exhibit improved generalization, despite the search traces being potentially noisy or suboptimal. However, relying on such imperfect traces can result in inefficient use of test-time compute. To address this, we propose *guided reinforced self-training* (Guided-ReST), a fine-tuning algorithm designed to improve the model's capability for effective search during inference. The key insight behind Guided-ReST is that optimal solutions can serve as valuable step-by-step landmarks to guide the model's search process. Based on this insight, we introduce a novel data generation method that seamlessly incorporates optimal solutions into the model's search procedure, enabling the generation of high-quality search traces. By fine-tuning the model on these search traces, we effectively distill improved search strategies into the model. Our method significantly enhances the search capabilities of language models on arithmetic reasoning and code self-repair tasks, including Countdown, CodeContests, and CodeForces. We release the source code at `https://github.com/snu-mllab/guided-rest`.

1 Introduction

Transformer-based language models have achieved remarkable success, demonstrating human-level performance across a wide range of natural language tasks, including conversation, code generation, and mathematical problem-solving [1, 31, 23, 12, 15, 26]. Their impressive performance is primarily attributed to auto-regressive training on massive, internet-sourced corpora. However, language models still encounter challenges in tasks that require complex planning and reasoning [18, 33]. To address these challenges, recent approaches have leveraged prompt-based strategies, enabling self-correction and planning via external symbolic search algorithms [34, 27, 38]. While these methods have shown success in specific contexts, their reliance on explicit prompting templates and manually engineered search structures restricts their flexibility and scalability.

More recent studies indicates that training language models to internalize the search process, rather than solely outputting the final solution, significantly improves their reasoning capabilities and allows performance to scale effectively with increased test-time compute [4, 9, 7]. An instructive example of test-time scaling is stream of search (SoS) [4], which trains a model to imitate search traces that encompass the complete decision-making process in finding optimal solutions through trial and error, including exploration and backtracking upon failure. This work demonstrates that models trained on search traces, despite being noisy and even suboptimal, exhibit superior generalization performance compared to those trained solely to imitate optimal solution via behavior cloning (BC) [22]. However,

*Corresponding author

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

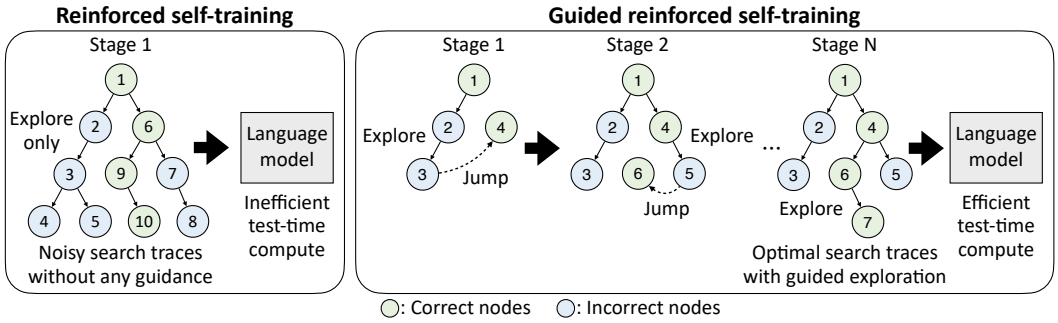

Figure 1: Overview of guided reinforced self-training, illustrating how search traces are generated through the progressive integration of optimal solutions during self-generation. The numbers indicate the order in which nodes are explored.

excessively noisy search traces can result in inefficient utilization of test-time compute, as the model may repeatedly explore irrelevant paths or perform redundant backtracking.

In this work, we examine how the sub-optimality of search traces affects test-time compute efficiency and propose a novel self-training algorithm, *guided reinforced self-training* (Guided-ReST), designed to improve search efficiency during inference. Our key insight is that while optimal solutions alone are insufficient for direct imitation, they can effectively serve as valuable landmarks to guide the search process. Inspired by jump-start reinforcement learning (JSRL) [32], we introduce a new data generation approach that progressively integrates optimal solutions into the model's search process, generating high-quality search traces, as illustrated in Figure 1. Subsequently, we fine-tune the model on these traces, thereby distilling more effective search strategies into the model. Finally, we further enhance performance by applying reinforcement learning (RL) fine-tuning.

We evaluate our method on the Countdown benchmark [4], a challenging arithmetic reasoning task that requires complex search. Our method achieves over a 10% accuracy improvement compared to baseline fine-tuning algorithms. Furthermore, it utilizes test-time compute more efficiently, reaching comparable performance while using less than half the number of tokens required by other baselines. Finally, we demonstrate the applicability of our method to a more realistic code self-repair task on the CodeContests and CodeForces benchmarks [14, 21].

2 Preliminaries

2.1 Stream of search

In this paper, we consider the problem of training a language model π_θ for tasks that require complex searching capabilities. Suppose that we have a training dataset $\mathcal{D}$ of question-solution pairs (q, S), where each solution consists of step-by-step reasoning $S = (s_1, \ldots, s_T)$. The most straightforward approach to train the model on this dataset is behavior cloning (BC) [22], which optimizes the model to generate the optimal solution using supervised learning:

$$\max_\theta \mathbb{E}_{(q,S)\sim\mathcal{D}} \left[\log \pi_\theta(S \mid q)\right].$$

However, prior studies have demonstrated that this approach struggles to generalize to unseen test examples [37, 11, 4, 39].

To address this, the steam of search (SoS) method introduces a novel training approach that emphasizes the search process itself rather than just the final solution [4]. Specifically, SoS formulates the problem as a tree search, exploring potential solutions through trial and error with operations until a successful solution is discovered. Each node in the tree represents a partial solution, and each edge represents a single reasoning step. The method represents primitive tree-search operations in language, including node generation, exploration, verification, and backtracking. For each question, it employs symbolic search algorithms such as depth-first search (DFS) and breadth-first search (BFS) to generate a search trace $Z = (z_1, \ldots, z_{T'})$, where each z_t denotes an individual search operation expressed in language.

The model is then trained to predict this search trace via supervised learning:

$$\max_{\theta} \mathbb{E}_{(q,Z)\sim\mathcal{D}} \left[\log \pi_\theta(Z \mid q)\right].$$

Note that in this framework, the optimal solution can also be viewed as a search trace, namely a path in the search tree. Additional details are provided in Appendix A.

2.2 Fine-tuning with self-generated data

While SoS demonstrates improved generalization by training a language model on search traces rather than optimal solutions, these traces are often noisy and suboptimal, resulting in imperfect alignment with the target task. To address this, SoS further fine-tunes the model using reinforcement learning techniques. One approach is reinforced self-training (ReST) [42, 6, 28], which fine-tunes the model on successful examples drawn from its self-generated responses via supervised learning:

$$\max_{\theta} \mathbb{E}_{q\sim\mathcal{D},Z\sim\pi_\theta(\cdot|q)} \left[\mathbb{1}_{R(Z|q)>\tau} \cdot \log \pi_\theta(Z \mid q)\right],$$

where R denotes the terminal reward function that evaluates the quality of the responses and τ denotes the threshold parameter.

Another approach is reinforcement learning (RL) fine-tuning [30, 17], which fine-tunes the model to directly maximize the reward while constraining the KL divergence from the reference model π_{ref} initialized from the pre-trained model:

$$\max_{\theta} \mathbb{E}_{q\sim\mathcal{D},Z\sim\pi_\theta(\cdot|q)} \left[R(Z \mid q) - \beta \cdot D_{\text{KL}}(\pi_\theta(\cdot \mid q) \parallel \pi_{\text{ref}}(\cdot \mid q))\right],$$

where $\beta > 0$ denotes the coefficient controlling the strength of the KL penalty term. This objective is typically optimized with proximal policy optimization (PPO) or group-relative policy optimization (GRPO) [25, 26].

2.3 Countdown benchmark

We use Countdown as the primary benchmark, following previous studies on searching with language models [38, 4]. Each problem consists of input numbers and a target number, and the objective is to combine the inputs using the four basic arithmetic operations to reach the target. For example, given a target number 26 and input numbers $[84, 2, 14, 15]$, a valid solution is $26 = 84 - 2 \times (14 + 15)$. Each arithmetic operation in this solution can be considered a single reasoning step. This task involves a high branching factor up to $O(k^2)$ at each node of the search tree, where k is the number of remaining inputs. To ensure a tractable difficulty level, we set the number of initial inputs to 4, consistent with the setup in Gandhi et al. [4]. Additional details are provided in Appendix B.

3 Methods

3.1 Motivation

SoS demonstrates that a language model trained on noisy, suboptimal search traces generalizes better than one trained only on clean, optimal solutions [4]. However, optimal solutions still offer valuable guidance during search. By providing the model with partial optimal solutions as hints, we can steer the model toward a reduced and more manageable search space, enabling more efficient exploration and significantly increasing the likelihood of success under a given token budget. Although optimal solutions are available only during training, the resulting trajectories provide high-quality supervision for fine-tuning.

This approach is closely aligned with the principle of jump-start reinforcement learning (JSRL) [32], which introduces a two-policy framework comprising a guide policy and an exploration policy. The guide policy, informed by prior knowledge or demonstrations, initiates the search process by steering the model toward promising regions of the solution space. The exploration policy then takes over and continues the search to reach the optimal solution, which significantly reduces the cost of exploration.

To investigate whether SoS can also benefit from this approach, we conduct a preliminary experiment. Given the optimal solution $S = (s_1, \ldots, s_T)$, we define a partial solution $(s_1, \ldots, s_t)$ consisting of

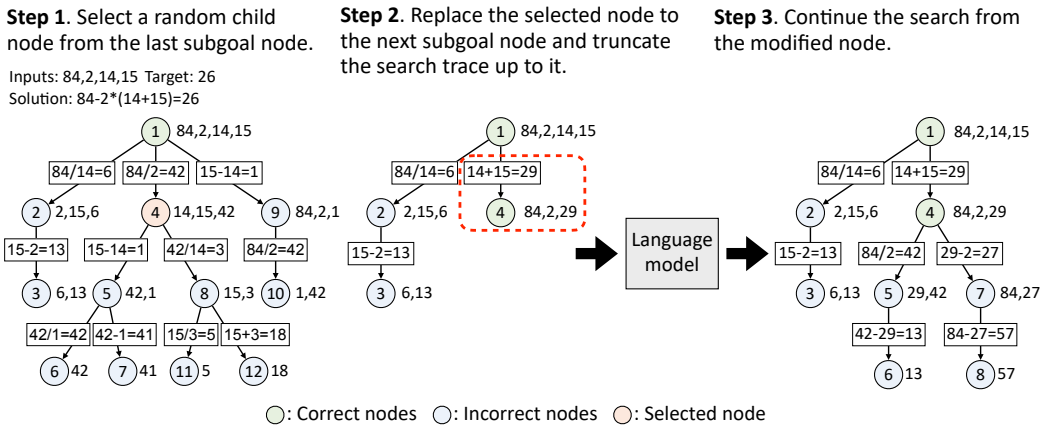

Figure 2: Overview of the subgoal augmentation algorithm. The numbers indicate the order in which nodes are explored. The red box highlights the modifications through node replacement.

Table 1: Accuracy and cross-entropy loss of search traces from partial solutions of different lengths.

Length	0	1	2	3
Accuracy (%) (↑)	55.96	85.28	95.27	100.00
Cross-entropy loss (↓)	0.0481	0.1245	0.2412	0.2539

the first t reasoning steps. We utilize this partial solution as an initial prompt to condition the model in generating a response. Table 1 shows the accuracy of search traces initialized from partial solutions of different lengths, evaluated over 10K training examples. The model successfully continues the search and discovers solutions when starting from partial solutions, even though it has not encountered these specific partial solutions during training. Moreover, increasing the amount of guidance by providing longer partial solutions greatly improves the accuracy of the resulting traces. This motivates us to leverage such high-quality, self-generated traces for fine-tuning, enabling the model to effectively internalize the successful search strategies.

However, naively distilling partial solutions as hints might negatively impact the model's performance. Table 1 shows the cross-entropy loss of successful search traces initialized from partial solutions of different lengths, demonstrating that traces from longer partial solutions results in higher loss values. This happens because longer partial solutions follow a distribution that differs significantly from the model's search behavior. Consequently, fine-tuning directly on these traces might cause substantial parameter updates, potentially degrading the model's search capabilities. Therefore, it is important to explore methods for effectively integrating optimal solutions into self-generated traces, ensuring that the resulting traces maintain both high likelihood and solution quality.

3.2 Guided reinforced self-training

In this subsection, we introduce a novel fine-tuning algorithm, called *guided reinforced self-training* (Guided-ReST), which seamlessly incorporates optimal solutions into the model's search process for data generation and effectively distills improved search behaviors into the model. The main idea is to leverage unsuccessful search attempts as context for each intermediate reasoning step within the optimal solution, mimicking the model's inherent search process and resulting in trajectories with high likelihood under the model. Before delving into our approach, we first introduce some notations. We define a subgoal node as any node along the optimal solution in the search tree. Thus, the optimal solution consisting of T reasoning steps contains T subgoal nodes.

Now, consider an unsuccessful search trace that has reached the $(t-1)$-th subgoal node but fails to navigate its subtree, and thus does not reach the t-th subgoal node. To transform it into a successful search trace, our method first randomly selects an unsuccessfully explored child node of the $(t-1)$-th subgoal node. Next, it replaces this child node with the correct k-th subgoal node and truncate the

Algorithm 1 AUGMENTSUBGOAL: Generating search traces via subgoal augmentation

Require: model π_θ, question q, search trace Z, optimal solution S, subgoal index t

if Z has already explored the t-th subgoal node of S **then**
  return Z
end if
Randomly select an explored child node from the $(t-1)$-th subgoal node of S
Replace the selected child node with the t-th subgoal of S
Truncate the search trace up to the modified node
Continue the search: $Z \leftarrow \pi_\theta(\cdot \mid q, Z)$
return Z

Algorithm 2 Guided reinforced self-training

Require: model π_{ref}, question q, optimal solution S

Initialize model parameters: $\pi_\theta \leftarrow \pi_{\text{ref}}$
for $i = 1, 2, \ldots,$ MAXITER **do**
  Generate a search trace: $Z \leftarrow \pi_\theta(\cdot \mid q)$
  for $t = 1, 2, \ldots, T$ **do**
    if Z is successful **then**
      break
    end if
    Add the t-th subgoal to the search trace: $Z \leftarrow \text{AUGMENTSUBGOAL}(\pi_\theta, q, Z, S, t)$
  end for
  Fine-tune the model via supervised learning: $\pi_\theta \leftarrow \text{TRAIN}(\pi_{\text{ref}}, q, Z)$
end for

search trace up to the modified node, as the subsequent search history becomes inconsistent with the updated subgoal. After the modification, the model continues the search from this augmented subgoal node. This algorithm, called *subgoal augmentation*, is summarized in Figure 2 and Algorithm 1. We iteratively run this algorithm until all subgoal nodes have been incorporated into the search trace. An example of the resulting search trace is provided in Appendix C. Finally, the model is fine-tuned on the successful search traces via supervised learning over multiple iterations. Algorithm 2 summarizes the overall training procedure.

A key advantage of Guided-ReST is its ability to explicitly teach the model where to backtrack during unsuccessful searches. Unlike ReST, which treats failed search traces as uninformative, or RL, which relies solely on terminal reward signals, it leverages optimal subgoal information to provide corrective feedback at failure points and guide the model to continue the search from more promising points. By systematically injecting the subgoal information to search traces, we expose the model to high-quality local corrections grounded in successful reasoning steps. This enables the model not only to generate correct answers but also to internalize structured reasoning strategies and the ability to recover from failure.

3.3 Operation-level RL fine-tuning

Building upon the fine-tuned model using Guided-ReST, we further improve its performance via RL fine-tuning using PPO. In standard PPO fine-tuning, the importance ratio is computed in a token-level Markov Decision Process (MDP). However, since our goal is to optimize the model's search strategy rather than token-level generation, we reformulate PPO in an operation-level MDP. Specifically, each action a_h is defined as a sequence of tokens $(a_{h,1}, \ldots, a_{h,L})$ that represents a single tree operation. Consequently, each state s_h is defined as the concatenation of the question and all previously executed operations. In this MDP formulation, the log importance ratio between the learner policy π_θ and the sampling policy $\pi_{\theta_{\text{old}}}$ is defined as

$$\log r_h(\theta) = \sum_{\ell=1}^{L} \left(\log \pi_\theta(a_{h,\ell} \mid a_{h,1:\ell-1}, s_h) - \log \pi_{\theta_{\text{old}}}(a_{h,\ell} \mid a_{h,1:\ell-1}, s_h)\right)$$

Algorithm 3 Guided reinforced self-training (episode-level)

Require: model π_{ref}, question q, optimal solution S, maximum turn limit T

Initialize model parameters: $\pi_\theta \leftarrow \pi_{\text{ref}}$
for $i = 1, 2, \ldots,$ MAXITER **do**
  Generate a multi-turn episode: $Z \leftarrow \pi_\theta(\cdot \mid q)$
  for $t = 1, 2, \ldots, T$ **do**
    if Z is successful **then**
      break
    end if
    # simplified subgoal augmentation
    Truncate the episode up to the t-th turn: $Z \leftarrow (S_1, E_2, \ldots, E_t, S_t)$
    Append user feedback containing the optimal solution $Z \leftarrow (S_1, E_2, \ldots, E_t, S_t, E'_{t+1})$
    Continue the episode: $Z \leftarrow \pi_\theta(\cdot \mid q, Z)$
  end for
  Fine-tune the model via supervised learning: $\pi_\theta \leftarrow \textsc{Train}(\pi_{\text{ref}}, q, Z)$
end for

Finally, we train the learner policy π_θ and the value function V_ϕ with the standard PPO objective:

$$\max_\theta \mathbb{E}_{(s_h,a_h)\sim\pi_{\theta_{\text{old}}}} \left[\min\left(r_h(\theta)A_h, \text{clip}\left(r_h(\theta), 1-\epsilon, 1+\epsilon\right)A_h\right)\right],$$

$$\min_\phi \mathbb{E}_{(s_h,a_h)\sim\pi_{\theta_{\text{old}}}} \left[\tfrac{1}{2}\left(V_\phi(s_h) - R_h\right)^2\right],$$

where $\epsilon > 0$ is the clipping coefficient. We compute the advantage A_h and return R_t using the Monte Carlo method instead of GAE [24], and remove the KL penalty term, following the recent practice of Yu et al. [40].

4 Application to episode-level search: code self-repair

So far, we have established our method on Countdown, a challenging yet formally defined task with a finite search space. In this section, we extend it to more realistic domains involving longer reasoning chains and more complex responses. Unlike Countdown, where the entire step-wise search trace fits within the context window, the increased output length and complexity make fine-grained tree search impractical. Therefore, we consider episode-level search, which generates complete solution attempts and iteratively refines them through multiple rounds of revision [20, 29, 10]. Specifically, given a question q, a search trace Z is represented as a multi-turn episode:

$$Z = (S_1, E_2, \ldots, E_T, S_T),$$

where S_t denotes a complete set of reasoning steps and the resulting solution generated by the model and E_t denotes user feedback containing verification results for the previous model's response and a revision instruction. The episode ends when the model's response passes verification or the maximum turn limit $T_{\max}$ is reached.

In this setting, we apply our method to the code self-repair task, where the model first generates an initial code solution and then iteratively refines it based on verification results from a public test set. The optimal solution is relatively easy to obtain because correctness can be automatically verified via code execution, making this task an effective testbed for evaluating our method. However, directly applying our method to this task still presents challenges, since subgoals are difficult to define within either the optimal solution or the search trace.

To address this issue, we employ a simplified, episode-level variant of Guided-ReST that progressively guides the model through user feedback augmented with the optimal solution as a hint. At each round t, the algorithm first truncates the current unsuccessful episode up to the t-th turn, then augments the subsequent user feedback E'_{t+1} with the optimal solution, and finally regenerates the remaining turns to complete the episode. The process repeats until the round reaches the maximum turn limit. As the round progresses, the number of turns augmented with the optimal-solution hint gradually increases, yielding progressively guided search traces. Finally, the model is fine-tuned on the successful search traces via supervised learning over multiple iterations. Algorithm 3 summarizes the overall training procedure. Additional details are provided in Appendix D.

5 Experiments

5.1 Setup

5.1.1 Countdown

We use Llama-3.2-1B-Instruct as the base model [5], with a maximum response length of 4K tokens.[2] For SoS, we generate search traces using heuristic-guided DFS and BFS over 500K training examples and fine-tune the base model for two epochs. For Guided-ReST, we generate a single search trace for each problem using 200K training examples and fine-tune the model for two epochs, repeating this procedure for three iterations. For PPO, we fine-tune the model on 200K training examples for two epochs. We adopt an outcome reward function that assigns 1 for success and 0 otherwise. Additional details are provided in Appendix E.

For evaluation, we follow the protocol of Gandhi et al. [4]. We construct 10K test examples for each of two settings: (1) seen targets, where the target numbers overlap with those in the training data but are paired with different input numbers, and (2) unseen targets, where the target numbers are entirely disjoint from the training data. We measure accuracy while varying the token budget.

We compare our method against BC, SoS, and the two fine-tuning methods introduced in Gandhi et al. [4]: (1) ReST and (2) PPO. Although the original SoS paper employs iterative advantage-induced policy alignment (APA) as the base RL algorithm [43], we find that PPO outperforms iterative APA. Unless otherwise specified, we conduct all RL fine-tuning experiments using operation-level PPO.

5.1.2 Code self-repair

We use Qwen2.5-7B-Instruct as the base model [36], with a maximum response length of 16K tokens. We build the training data using PrimeIntellect's SYNTHETIC-1 [16], which was curated from APPS, CodeContests, and TACO [8, 14, 13]. We select problems that have at least one ground-truth solution, resulting in 16K problems. For Guided-ReST, We generate eight episodes per problem, each with a maximum of four turns, and fine-tune the model for two epochs, repeating this procedure for three iterations. We compute the loss only on the last model response, following the practice of Snell et al. [29]. Additional details are provided in Appendix E.

For evaluation, we consider two code benchmarks: (1) CodeContests and (2) CodeForces [14, 21]. CodeContests consists of 165 problems and represents a near-distribution evaluation, as our training data was partially curated from the same source. On the other hand, CodeForces consists of 408 more recent problems, providing a more challenging out-of-distribution benchmark. We measure accuracy by extracting code from the model's response and executing it on private test cases.

We compare our method against the base model and ReST. We do not perform RL fine-tuning due to limited computational resources. Instead, we evaluate performance using pass@k accuracy, which is a reliable proxy for RL fine-tuning [41].

5.2 Countdown results

Figure 3 shows the performance of each method across different token budgets on Countdown, where responses are sampled using greedy decoding. Our method significantly outperforms SoS and the baseline fine-tuning methods. At the maximum token budget, our method achieves 87% accuracy on both seen and unseen targets, outperforming the second-best PPO by more than 10%. It is important to note that the improved performance on unseen targets indicates that our method does not merely memorize the optimal solutions provided as guidance. Instead, it internalizes a more effective search strategy, leading to stronger generalization. Moreover, our method achieves comparable accuracy to the second-best PPO while consuming only about 50% of its tokens, demonstrating a more efficient utilization of test-time compute. BC achieves less than 40% accuracy on both seen and unseen targets, indicating substantially weaker generalization performance compared to search-based models.

[2]We choose the Llama-3 model for Countdown due to its efficient tokenization of integers.

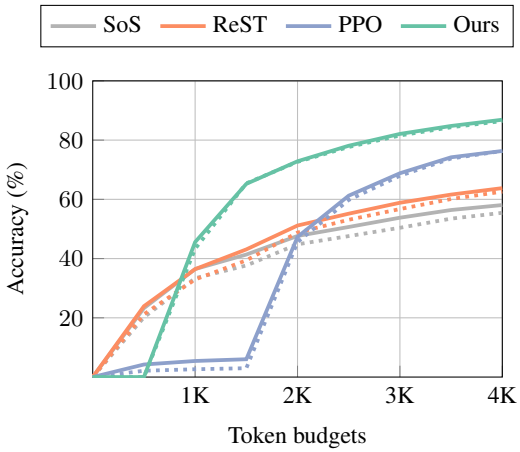

Figure 3: Accuracy of our method and baselines on Countdown. Solid lines represent seen targets, and dotted lines represent unseen targets.

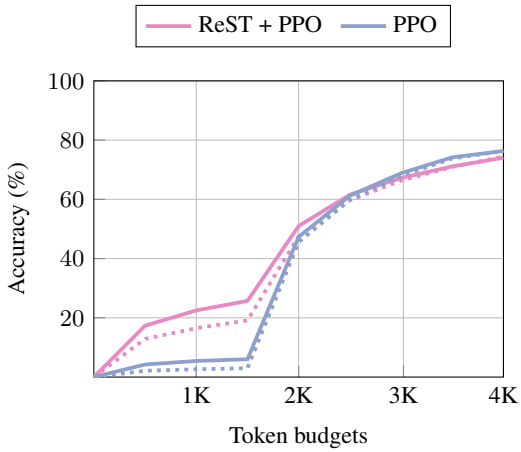

Figure 4: Accuracy of PPO initialized with ReST on Countdown. PPO does not benefit from ReST in the high-budget regime.

Table 2: Pass@k accuracy of ReST and Guided-ReST on Countdown (32 samples per problem).

Method	Seen target						Unseen target					
	1	2	4	8	16	32	1	2	4	8	16	32
SoS	55.3	63.8	69.6	73.3	75.5	77.1	53.2	62.3	68.7	73.0	75.6	77.5
ReST	62.3	67.7	71.3	73.6	75.3	76.6	60.8	66.7	70.8	73.5	75.5	77.0
Guided-ReST	**62.7**	**75.3**	**84.6**	**90.8**	**94.7**	**96.8**	**61.0**	**74.1**	**83.8**	**90.3**	**94.3**	**96.8**

5.3 Impact of Guided-ReST on RL fine-tuning

Our method leverages both Guided-ReST and PPO for fine-tuning. This naturally raises the question of whether PPO can achieve similar synergy with standard ReST, which does not leverage optimal-solution guidance during data generation. Figure 4 presents the accuracy of PPO fine-tuning initialized with ReST. We observe that PPO does not benefit from ReST, but achieves significant improvements when combined with Guided-ReST.

To examine why Guided-ReST exhibits stronger synergy with PPO, we evaluate the pass@k accuracy. Table 2 presents the pass@k accuracy of ReST and Guided-ReST on Countdown, where responses are sampled at a temperature of 1.0. ReST outperforms SoS at pass@1 but shows almost no difference at pass@32. Recent work suggests that RL mainly shifts probability mass from already-likely correct responses toward top-ranked ones [41]. Consequently, since ReST and SoS exhibit similar coverage of correct candidates, PPO initialized with ReST yields performance comparable to that initialized with SoS. In contrast, Guided-ReST achieves a similar pass@1 accuracy to ReST but delivers much larger accuracy gains as k increases. This broader coverage provides PPO with more correct candidates to amplify, leading to significant performance improvements.

5.4 Analysis of operation-level MDP

We investigate the effectiveness of the operation-level MDP formulation for RL fine-tuning introduced in Section 3.3. Figure 5 shows the performance of our method under the standard token-level MDP. We find that fine-tuning the model under the operation-level MDP not only improves performance from 83% to 87% at the maximum token budget but also substantially enhances test-time compute efficiency. Compared to the token-level MDP, our approach achieves approximately 2$\times$ higher token efficiency in the low-budget regime and about 1.5$\times$ higher efficiency in the high-budget regime. This result highlights the importance of defining an MDP aligned with the optimization objective.

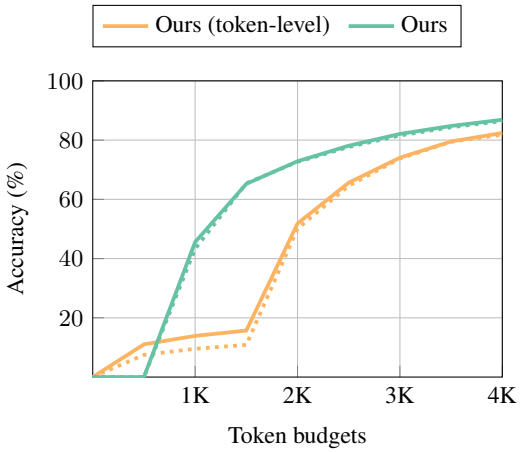

Figure 5: Accuracy of our method when trained under the token-level MDP on Countdown.

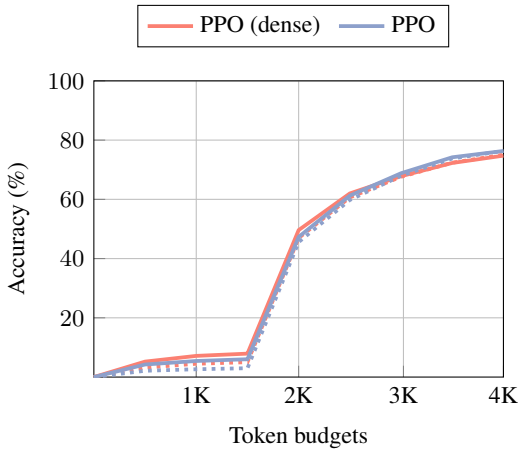

Figure 6: Accuracy of PPO trained with a dense reward on Countdown.

Table 3: Pass@k accuracy of ReST and Guided-ReST on code self-repair (128 samples per problem).

Method	CodeContests						CodeForces					
	1	2	4	8	16	32	1	2	4	8	16	32
Base	4.5	7.5	11.3	15.7	20.6	25.8	5.5	8.5	12.5	17.2	22.4	27.7
ReST	9.4	13.1	17.0	21.3	25.9	30.4	**9.7**	14.2	19.3	24.8	30.5	35.9
Guided-ReST	**10.5**	**14.8**	**19.4**	**24.1**	**28.9**	**33.9**	**9.7**	**14.5**	**20.2**	**26.2**	**32.0**	**37.6**

5.5 RL fine-tuning with dense reward

Our method leverages subgoal guidance derived from optimal solutions during data generation. As an alternative, one can directly perform RL fine-tuning using subgoal-based rewards. Given an optimal solution $S = (s_1, \ldots, s_T)$ and a search trace $Z = (z_1, \ldots, z_{T'})$, we define the subgoal reward as

$$R_{\text{subgoal}}(z_{1:t'} \mid q) = \begin{cases} 0.25 & \text{if there exists } s_t \text{ such that } z_{t'} \text{ reaches } s_t \\ 0 & \text{otherwise,} \end{cases}$$

We fine-tune the SoS model using PPO with a dense reward that combines the subgoal and outcome reward functions.[3] To prevent reward hacking, the subgoal reward is applied only once per subgoal.

Figure 6 presents the performance of PPO with the dense reward. The dense reward has little effect on overall performance and even causes a slight degradation in the high-budget regime. These results underscores the importance of Guided-ReST in conditioning the model before the RL phase.

5.6 Code self-repair results

As observed in the Countdown experiment in Section 5.3, Guided-ReST achieves similar pass@1 accuracy to ReST but yields higher pass@k as k increases, which translates into greater improvements during PPO fine-tuning. To examine whether this effect generalizes beyond Countdown, we conducted the same analysis on the code self-repair task.

Table 3 presents the pass@k accuracy of ReST and Guided-ReST on CodeContests and CodeForces. Guided-ReST consistently achieves higher accuracy than ReST, with the gap between them widening as k increases. Note that ReST also outperforms the base model, as the base model is general-purpose rather than task-specific. The overall improvement remains moderate compared to Countdown, likely due to fewer training examples, incomplete subgoal augmentation, and weaker search capabilities.

[3]The total reward ranges from 0 to 1.5, as each Countdown problem contains two non-trivial subgoals.

6 Related work

Searching with language models Yang et al. [37] first introduce the concept of training a sequence model to imitate expert MCTS policy search traces, primarily to enhance imitation learning. More recent studies have shifted focus toward directly improving search capabilities by training language models on these traces and fine-tuning them with self-generated data [11, 4]. In contrast to these prior studies, which apply generic self-improvement algorithms not tailored for search [42, 6], our work analyzes the relationship between the quality of self-generated data and the efficiency of test-time compute, and introduces a self-improvement algorithm specifically optimized for search efficiency. Several studies have also explored sequential revision and episode-level search [35, 20, 29], but these do not address improvements in test-time compute efficiency.

Fine-tuning on self-generated data Anthony et al. [2] first introduce the idea of iteratively distilling self-generated data into neural networks, generating high-quality trajectories using an expert MCTS policy and imitating them to improve a neural network-based policy. Recent studies have extend this idea to fine-tune language models on self-generated data for various tasks, including mathematical reasoning, question answering, and machine translation [19, 42, 6], although these approaches are not specifically tailored for search. The work most closely related to ours is that of Chang et al. [3], which employs a stronger guide model alongside a weaker base model to generate data and fine-tune the base model using PPO. Their method, however, is limited to a single round of interaction between the guide and base models. Our approach performs multiple rounds of guide-base interactions, enabling richer and more iterative supervision.

7 Conclusion

We introduce Guided-ResT, a self-training algorithm designed to improve the search efficiency of language models. By incorporating optimal solution as guidance for generating high-quality search traces, our method enables models to internalize more effective search strategies. Experiments on the Countdown and code self-repair tasks demonstrate that our method not only enhances accuracy but also scales robustly with increased test-time compute, outperforming existing fine-tuning approaches.

Our work is not without limitations. It assumes access to optimal solutions, which may not always be available in practice. Nevertheless, this assumption can be relaxed by leveraging solutions generated by a sufficiently capable model. Once such a model is available, a promising direction is to develop a collaborative framework in which a stronger model provides guidance when the base model fails to make progress, and this interaction is subsequently distilled into the base model through additional training. This teacher-student framework facilitates the transfer of effective search strategies from the teacher to the student model.

Another limitation lies in the assumption of an oracle in Countdown that can precisely identify the first incorrect step and backtrack to that point. Although such an oracle provides a clear and reliable signal for targeted correction, this assumption becomes unrealistic when replaced with a language model, particularly standard instruction-following models that still struggle to recognize reasoning errors or perform localized backtracking. A promising direction for future work is to extend our approach to reasoning models that already exhibit emerging backtracking and self-correction capabilities [7].

Acknowledgements

This work was supported in part by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS2020-II200882, (SW STAR LAB) Development of deployable learning intelligence via self-sustainable and trustworthy machine learning, No. RS-2022-II220480, Development of Training and Inference Methods for Goal-Oriented Artificial Intelligence Agents, No. RS-2021-II211343, Artificial Intelligence Graduate School Program (Seoul National University)), and by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2024-00354036). This research was supported by a grant from KRAFTON AI. This material is based upon work supported by the Air Force Office of Scientific Research under award number FA2386-23-1-4047. Hyun Oh Song is the corresponding author.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. In *NeurIPS*, 2017.

[3] Jonathan D Chang, Kiante Brantley, Rajkumar Ramamurthy, Dipendra Misra, and Wen Sun. Learning to generate better than your llm. *arXiv preprint arXiv:2306.11816*, 2023.

[4] Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah D Goodman. Stream of search (sos): Learning to search in language. *arXiv preprint arXiv:2404.03683*, 2024.

[5] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[6] Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, et al. Reinforced self-training (rest) for language modeling. *arXiv preprint arXiv:2308.08998*, 2023.

[7] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[8] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, et al. Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938*, 2021.

[9] Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

[10] Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, et al. Training language models to self-correct via reinforcement learning. In *ICLR*, 2025.

[11] Lucas Lehnert, Sainbayar Sukhbaatar, Paul Mcvay, Michael Rabbat, and Yuandong Tian. Beyond a*: Better planning with transformers via search dynamics bootstrapping. *arXiv preprint arXiv:2402.14083*, 2024.

[12] Raymond Li, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, LI Jia, Jenny Chim, Qian Liu, et al. Starcoder: may the source be with you! *Transactions on Machine Learning Research*, 2023.

[13] Rongao Li, Jie Fu, Bo-Wen Zhang, Tao Huang, Zhihong Sun, Chen Lyu, Guang Liu, Zhi Jin, and Ge Li. Taco: Topics in algorithmic code generation dataset. *arXiv preprint arXiv:2312.14852*, 2023.

[14] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 2022.

[15] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

[16] Justus Mattern, Sami Jaghouar, Manveer Basra, Jannik Straube, Matthew Di Ferrante, Felix Gabriel, Jack Min Ong, Vincent Weisser, and Johannes Hagemann. Synthetic-1: Two million collaboratively generated reasoning traces from deepseek-r1, 2025. URL `https://www.primeintellect.ai/blog/synthetic-1-release`.

[17] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. In *NeurIPS*, 2022.

[18] Vishal Pallagani, Bharath Muppasani, Keerthiram Murugesan, Francesca Rossi, Biplav Srivastava, Lior Horesh, Francesco Fabiano, and Andrea Loreggia. Understanding the capabilities of large language models for automated planning. *arXiv preprint arXiv:2305.16151*, 2023.

[19] Stanislas Polu, Jesse Michael Han, Kunhao Zheng, Mantas Baksys, Igor Babuschkin, and Ilya Sutskever. Formal mathematics statement curriculum learning. *arXiv preprint arXiv:2202.01344*, 2022.

[20] Yuxiao Qu, Tianjun Zhang, Naman Garg, and Aviral Kumar. Recursive introspection: Teaching language model agents how to self-improve. In *NeurIPS*, 2024.

[21] Shanghaoran Quan, Jiaxi Yang, Bowen Yu, Bo Zheng, Dayiheng Liu, An Yang, Xuancheng Ren, Bofei Gao, Yibo Miao, Yunlong Feng, et al. Codeelo: Benchmarking competition-level code generation of llms with human-comparable elo ratings. *arXiv preprint arXiv:2501.01257*, 2025.

[22] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011.

[23] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

[24] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *ICLR*, 2016.

[25] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[26] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, YK Li, Yu Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[27] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *NeurIPS*, 2023.

[28] Avi Singh, John D Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J Liu, James Harrison, Jaehoon Lee, Kelvin Xu, et al. Beyond human data: Scaling self-training for problem-solving with language models. *arXiv preprint arXiv:2312.06585*, 2023.

[29] Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling test-time compute optimally can be more effective than scaling llm parameters. In *ICLR*, 2025.

[30] Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. In *NeurIPS*, 2020.

[31] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[32] Ikechukwu Uchendu, Ted Xiao, Yao Lu, Banghua Zhu, Mengyuan Yan, Joséphine Simon, Matthew Bennice, Chuyuan Fu, Cong Ma, Jiantao Jiao, et al. Jump-start reinforcement learning. In *ICML*, 2023.

[33] Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. On the planning abilities of large language models - a critical investigation. In *NeurIPS*, 2023.

[34] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *ICLR*, 2023.

[35] Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khashabi, and Yejin Choi. Generating sequences by learning to self-correct. In *ICLR*, 2023.

[36] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

[37] Mengjiao Sherry Yang, Dale Schuurmans, Pieter Abbeel, and Ofir Nachum. Chain of thought imitation with procedure cloning. In *NeurIPS*, 2022.

[38] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *NeurIPS*, 2023.

[39] Tian Ye, Zicheng Xu, Yuanzhi Li, and Zeyuan Allen-Zhu. Physics of language models: Part 2.2, how to learn from mistakes on grade-school math problems. *arXiv preprint arXiv:2408.16293*, 2024.

[40] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

[41] Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *arXiv preprint arXiv:2504.13837*, 2025.

[42] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. In *NeurIPS*, 2022.

[43] Banghua Zhu, Hiteshi Sharma, Felipe Vieira Frujeri, Shi Dong, Chenguang Zhu, Michael I Jordan, and Jiantao Jiao. Fine-tuning language models with advantage-induced policy alignment. *arXiv preprint arXiv:2306.02231*, 2023.

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes] , [No] , or [NA] .
- [NA] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for NA).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will be asked to also include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While "[Yes] " is generally preferable to "[No] ", it is perfectly acceptable to answer "[No] " provided a proper justification is given (e.g., "error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering "[No] " or "[NA] " is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We clearly explain our paper's contribution in the abstract and introduction.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We clearly explain the limitations in the conclusion.

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

   Justification: Our paper does not include theoretical results.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide all the information (*e.g.*, hyperparameters) in the supplementary material.

   Guidelines:

   - The answer NA means that the paper does not include experiments.

- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We provide a link to the source code in the abstract.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).

- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We specify all details in the experiment section and supplementary material.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We run experiments with multiple random seeds (32 to 128) and report average results for those with high variance.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
   - If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provide the computation resources in the supplementary materials.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.

- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have carefully reviewed the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss potential societal impacts in the Supplementary materials.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: Our work poses no such risks, as it focuses on arithmetic and code self-repair tasks.

    Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We provide the license of the models we used in the supplementary material.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We provide a link to the source code in the abstract.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

Justification: We does not involve crowdsourcing.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: We does not involve crowdsourcing.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: We use LLMs only for writing.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.