IMPROVING TEST-TIME SEARCH WITH BACKTRACKING AGAINST IN-CONTEXT VALUE VERIFIERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Test-time verifiers are useful for solving reasoning problems as they guide the reasoning chain towards valid solutions, thus improving performance. However, naïvely conducting search with approaches like Best-of-N sampling with these verifiers is often inefficient, requiring the generation of multiple overlapping partial solutions and incorrect completions to arrive at a correct outcome. This raises the question: can we reduce the total amount of computation by sharing information across multiple attempts at solving a given reasoning problem? In this paper, we build an approach of combining process verifiers that predict likelihoods of success *per step* with preemptive backtracking to maximize performance per generated token. To do this, a process reward model (PRM) can be used to identify the problematic step in a solution trace, allowing the model to selectively resample from the problematic step of a solution. This approach can significantly reduce the amount of computation by leveraging partial reasoning traces from previous revisions. We also introduce in-context process supervision, where the verifier is conditioned on the history of past revisions. This reduces uncertainty in the verification decisions and improves the verifier's confidence with each round of backtracking. This framework for iterative backtracking, leveraging in-context process supervision, enables an effective trade-off between inference and performance.

1 INTRODUCTION

One promising direction for enhancing quality of responses in reasoning problems is the strategic use of test-time computation, where a model is given an inference compute budget that it could leverage to improve its solution quality. Prior work such as Snell et al. (2024); Charniak & Johnson (2005) and Cobbe et al. (2021) explore different mechanisms for solution generation given a fixed generation budget, resulting in linear search algorithms such as BofN, Beam Search, and Look-Ahead search. A major drawback of these linear search approaches is their compute efficiency. On hard problems, small but consequential mistakes made at intermediate steps can lead to a cascade of errors, rendering the entire sequence incorrect. Traditional search methods, which operate in parallel and start from the beginning of the sequence, fail to efficiently handle these errors, often wasting inference-time compute on parts of the solution that are already correct. Can we more efficiently leverage inference-time compute for error recovery?

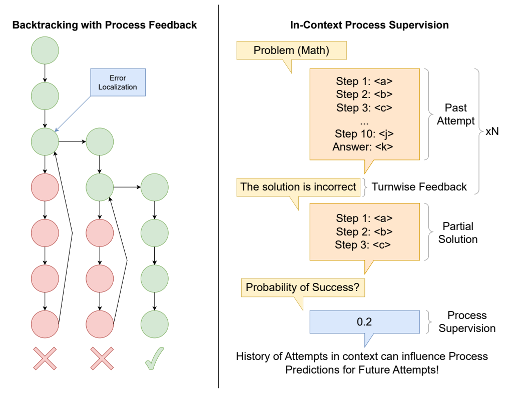


Figure 1: **(Left) An illustration of utilizing a value-based verifier for backtracking.** Backtracking with a value-based verifier allows for revising a potential incorrect reasoning chain. We use an error localization criterion to identify an appropriate step to regenerate the solution from, and continue to iteratively revise the solution until a stopping criterion is met. **(Right) Leveraging In-Context Process Supervision.** This approach conditions on prior attempts to reduce the uncertainty of process reward estimates on future attempts.

In this paper, we propose **backtracking-based iterative refinement** for improving LLM solutions. Instead of generating multiple complete solutions with many redundant tokens, our method generates a single solution, identifies the problematic steps in the solution trajectory, and resamples only the parts of the sequence that need correction. This adaptive approach not only reduces the computational burden but also allows for more targeted revisions, effectively correcting uncorrelated errors and improving the overall solution quality.

Our method leverages advantage estimates, computed using a process-based verifier (PRMs), to predict the points in the reasoning chain where the model is most likely to make mistakes. These advantage estimates assess the relative importance of each step and guide the backtracking process to the step that contributes least to the solution quality. The backtracking algorithm then selectively resamples the completions from that step. These set of targeted revisions allow for compute to be spent more efficiently, leading to higher-quality solutions with fewer revisions. We also establish a stopping criterion in this framework for early termination of backtracking after a solution has been found.

Ultimately the success of backtracking relies on how accurate the PRM is on intermediate reasoning steps. In this paper, we also propose various methods to improve the verifier accuracy. A major concern with use of verifiers for search is distribution shift. Verifiers are trained on static data that is either human-labeled or generated using models that are potentially different than the ones used during the inference. To temper these distribution shift issues, we employ on-policy verifier fine-tuning and label balancing, and propose a novel method to smooth the verifier's outputs to account for estimation errors. We additionally introduce in-context process supervision which allows the PRM to condition on prior attempts at a problem to reduce the uncertainty of the supervision on future attempts. Conditioning on a history of past revisions, the verifier is able to identify failure modes in past attempts and subsequently avoid rating them highly if they are present in future attempts. By providing a framework for sequentially identifying errors and revising them, our approach offers a more scalable and effective solution for test-time inference in LLMs, making it particularly suited for tasks that are more difficult or longer horizon. We demonstrate the computational efficiency of our approach, showing that it significantly improves the test-time compute tradeoff with respect to the number of generated tokens vs. the accuracy of the generated solution.

2 Preliminaries and Notation

A **process reward model (PRM)** assesses the validity of intermediate steps taken during the reasoning process, providing feedback on whether an intermediate step makes progress towards the solution. One instantiation of the PRM is a **value function** or **value verifier**, learned through Monte-Carlo (MC) regression Wang et al. (2023); Snell et al. (2024), where the value at each step is supervised with the Monte-Carlo return-to-go, $\mathcal{R}^{\mathcal{D}}(s,a)$, which estimates the probability of success of a rollout from a particular state s and action a. Here, $\mathcal{D}$ represents the static dataset used to compute the target return-to-go values. The Monte-Carlo return-to-go is computed as the sum of future discounted rewards, $\mathcal{R}^{\mathcal{D}}(s,a) = \sum_{t=i}^{t=T} \gamma^t r_t$, where γ is the discount factor, and the sum is taken over the trajectory from the current time step i to the end of the episode. We assume the reward function to be sparse, i.e., it is 0 for each intermediate step in the reasoning problem. At the terminal step, if the predicted answer matches the ground truth answer, a reward of 1 is provided for this final step. The loss function is the divergence of the estimated value from the dataset Monte-Carlo return-to-go:

$$\mathcal{L}_{\mathcal{Q}} = \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[\mathcal{L}_{KL}\left(Q(s,a), \mathcal{R}^{\mathcal{D}}(s,a)\right)\right] \tag{1}$$

One instantiation of the divergence is soft binary cross-entropy, where each step $(\mathbf{s}_t, \mathbf{a}_t)$ has a target $y_t \in [0,1]$:

$$\mathcal{L}_{\text{process}} = -\sum_t \left[y_t \log \hat{y}_t + (1-y_t)\log(1-\hat{y}_t)\right], \tag{2}$$

where $\hat{y}_t = \mathcal{R}^{\mathcal{D}}(s,a)$ is the predicted probability that the step $(\mathbf{s}_t, \mathbf{a}_t)$ is correct.

In contrast, an **outcome reward model (ORM)** only evaluates the correctness of the final answer, disregarding individual reasoning steps. Let $\mathbf{o}$ represent the final outcome of the reasoning process. We can similarly learn an ORM through binary classification, where the final outcomes $\mathbf{o}$ are labeled with binary labels $y_{\text{outcome}} \in \{0, 1\}$:

$$\mathcal{L}_{\text{outcome}} = -\sum_i \Big[y_{\text{outcome},i} \log \hat{y}_{\text{outcome},i} + (1 - y_{\text{outcome},i}) \log(1 - \hat{y}_{\text{outcome},i}) \Big] \tag{3}$$

where $\hat{y}_{\text{outcome},i} = R_{\text{outcome}}(\mathbf{o}_i)$ is the predicted probability that the final outcome $\mathbf{o}_i$ is correct.

Linear search in test-time inference: Test-time inference often requires efficient search strategies to navigate the potential solution space. We define **linear search algorithms** as those that operate with a fixed compute budget and a predetermined width of potential completions. Prior work Snell et al. (2024) leverages linear search algorithms like Beam Search and Best-of-N to find near-optimal solutions during inference.

In Best-of-N, n candidate solutions are generated and the solution with the highest evaluation score is selected:

$$\hat{y} = \arg \max_{y_i \in \{y_1, y_2, \ldots, y_n\}} S(y_i) \tag{4}$$

Here, S, is either an outcome or process reward or score function, and $\hat{y}$ is the selected output.

In beam search, multiple solution paths are expanded in parallel. At each step t, the algorithm retains the top k candidates based on their scores $S(y_{1:t})$:

$$\mathcal{B}_t = \text{Top-}k\left(\{y_{1:t-1} \cdot y_t \mid y_t \in \mathcal{Y}\}, S(y_{1:t})\right) \tag{5}$$

where $\mathcal{B}_t$ represents the set of the top k sequences at step t, $\mathcal{Y}$ is the set of all possible tokens, and $S(y_{1:t})$ is the evaluation score of the sequence $y_{1:t}$. These methods provide varying trade-offs between computational efficiency and search accuracy, helping to explore the reasoning space effectively during inference.

Linear Search methods can be effective in parallel sampling scenarios, where sequential revisions of a solution are not possible, enabling the discovery of better solutions than single-shot sampling from the model. A **non-linear search algorithm**, in contrast, can adaptively allocate inference time compute. This can allow for more inference time compute to be spent on portions of the problem that are harder to get right. To build a non-linear search algorithm, we will formalize our intuitions in a multi-step single-turn Markov Decision Process (MDP), which we describe next.

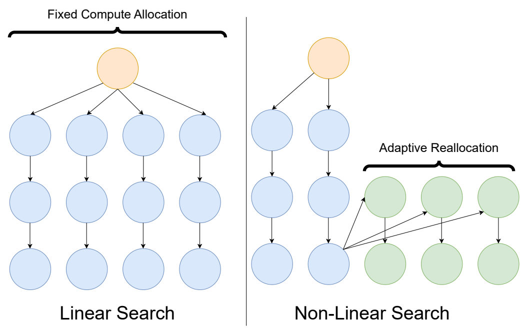


Figure 2: **Linear vs Non-Linear Search** Linear search assumes a fixed compute allocation during inference time. However, with Non-Linear search, we can adaptively allocate inference time compute to parts of the reasoning chain that require more computation.

Multi-step, Single-turn Markov Decision Process (MDP) for Reasoning: A reasoning chain in test-time inference can be conceptualized as a multi-step Markov Decision Process (MDP). We formally define a reasoning chain as a trajectory with a bounded horizon H, represented as $(s, a, r, t)_0^H$. These trajectories originate from a dataset of prompts x and responses y, which is decomposed into several semantic steps $a_1, a_2, \cdots, a_t$ that, when concatenated, reconstruct the original response y.

We formally define our MDP as:

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, R, \gamma, \rho_0), \tag{6}$$

where:

- $\mathcal{S}$: The state s_i at timestep i is the current prompt x and the set of previous semantic steps $a_{0 \ldots i-1}$.
- $\mathcal{A}$: The action a_i is the next semantic step in the reasoning chain.

- $\mathcal{T}$: The transition dynamics function is concatenation: $P(s'|s,a) = \text{concat}(s,a)$.
- R: The reward $r = R(s,a)$, evaluates the quality or relevance of the current semantic step.
- γ: The discount factor.
- ρ_0: The initial state distribution, representing the distribution of initial prompts x_i.

3 Backtracking for Sequential Improvement

To motivate our backtracking approach, we first consider a didactic problem where an LLM has the goal of generating a repeating sequence of digits from 0 to 9. The task involves producing a correct sequence over $N = 1500$ tokens, evaluated by exact match. This setup is depicted in Figure 3. Suppose a base model performs well overall but consistently struggles with generating the digit **6** when $t \equiv 6 \mod 10$, outputting 6 only 5% of the time and incorrectly outputting 1 otherwise. The base policy is defined as:

$$\pi_{\text{base}}(a_t \mid s_t) = \begin{cases} \Pr(a_t = 6) = 0.05, & \text{if } t \equiv 6 \mod 10, \\ \Pr(a_t = 1) = 0.95, & \text{if } t \equiv 6 \mod 10, \\ \Pr(a_t = d \neq 6) = 1, & \text{if } t \equiv d \mod 10. \end{cases}$$

With over 150 opportunities to generate the digit 6, this model outputs it correctly only ≈ 8 times, leading to 142 errors that compound across the sequence.

Linear Search is not enough: Linear search algorithms like Best-of-N struggle in this scenario, and would need to generate $\mathbb{E}[N] = 20^{150}$ tokens, which is computationally infeasible. For these types of uncorrelated errors, the expected number of tokens required to produce a fully correct sequence increases *exponentially* with the number of errors. In contrast, *non-linear search algorithms*, such as backtracking, can use compute resources more efficiently. By allowing the model to retry from before an erroneous state, it can iteratively correct mistakes without regenerating correct parts of the sequence. For example, if an oracle identifies the error location in a solution, backtracking to it and applying Best-of-N from there can fix a single mistake with $\mathbb{E}[N] = 20$. Repeating this process across 150 errors yields an expected sample complexity of $\mathbb{E}[N] = 20 \times 150$, which is *linear* in the number of errors. This highlights the need for non-linear search methods like backtracking, which avoid wasted computation. We will now devise a framework for effective backtracking.

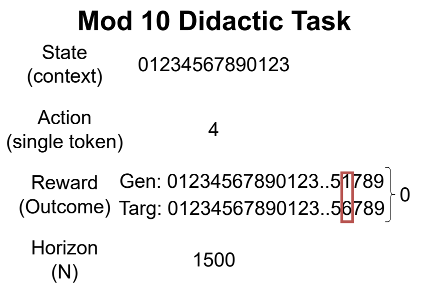


Figure 3: **Didactic Task**: Mod 10 sequence generation. The state is the context or the set of characters generated so far. The action is a single character that is generated. The horizon of generation is 1500 characters. The reward is provided at the outcome level, whether the sequence matches the target sequence.

3.1 Backtracking Framework for Solution Revision

In this work, we develop an effective framework for non-linear search via backtracking. Doing so requires answering the following key questions: **(1)** How can we identify problematic parts of a solution and revise them? **(2)** Can we leverage the history of sequential revisions to better guide search and identify problematic parts of a solution? **(3)** What are practical considerations to maximize efficacy and robustness of such a backtracking framework? In the following discussion, we answer each of these questions.

Localizing incorrect steps with PRMs. The first key component of the framework is identifying where in the solution a mistake appears. One natural choice for this is the **advantage function**, $A^\pi(s,a)$, which measures the difference in the expected success of a particular action at a particular state (i.e., step of a reasoning chain) compared to a baseline $V^\pi(s)$, or the expected success of actions queried from a proposal distribution or base policy. More formally, this is defined as:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s) \tag{7}$$

Intuitively, here the baseline can be viewed as a form of calibration/normalization, where the advantage function not only considers the expected success rate (PRM, $Q^{\pi}(s,a)$) but the success rate relative to the performance of the proposal distribution or policy.

When the advantage function is low for a step, this means that this step is largely reducing the probability of arriving at the right answer. Therefore, backtracking and resampling may be desirable as a better step can be obtained from the proposal distribution. This motivates using the minimum advantage step within a trajectory τ to revise from:

$$i_{revise} = \min_i A(s_i, a_i), i \in \{1, \dots, H\} \tag{8}$$

where H is the horizon of a trajectory τ.

Modeling two different functions, $Q^{\pi}(s,a)$ and $V^{\pi}(s)$ is undesirable due to computational inefficiencies in training and querying both functions. One thing we can leverage is that the dynamics of the underlying MDP (as seen in Section 2) is deterministic. Thus, we can choose to model only a Q-value function (PRM), $Q^{\pi}(s,a)$ and use it to compute $V^{\pi}(s)$. This can be done by querying the function at the previous state, computing the advantage as the value difference between subsequent steps within the trajectory.

$$\begin{gathered} A^{\pi}(s', a') = Q^{\pi}(s', a') - Q^{\pi}(s, a), \text{ where } s' = \textit{concat}(a, s), \\ i_{\text{revise}} = \min_i Q^{\pi}(s_i, a_i) - Q^{\pi}(s_{i-1}, a_{i-1}) \end{gathered} \tag{9}$$

This reformulation allows us to additionally view the advantage as a measure of progress, or how much an action contributes to the success of a trajectory.

Completing revisions and stopping criterion. Once the problematic step i_{revise} has been identified, we can re-sample the entire suffix of the solution, conditioned on the partial solution before the minimum advantage step with a linear search algorithm such as Best-Of-N sampling or Beam Search. The value function evaluated at the final step serves the role of an outcome-level reward signal. Therefore, we can use the value-function for this search procedure that seeks to generate the full revised solution. Additionally, we can do this process for multiple rounds, allowing for subsequent revisions from the previously modified reasoning chain(s). We perform this backtracking process a maximum of M times, ensuring that the revision process does not continue indefinitely and we can control the sample budget.

3.2 In-Context Value Verifiers

So far, we built a simple algorithm to identify problematic steps in a reasoning chain and find alternative completions. While effective, this approach is fairly naïve by itself: most of the efficiency gains so far come only from prefix sharing. This means that if the base model were prone to repeatedly making the same underlying mistake, but it does so in several diverse natural language phrases, our approach may not be the most effective at improving token efficiency. One way to address this problem is ensure that the value function used for scoring completions is made aware of the mistakes made by the LLM in previous search attempts. To do so, in this section, we will extend verifiers to include some notion of "state" of the search process. This can be done by conditioning the verifier on a linearization of the entire search procedure so far, which we denote as in-context verifiers.

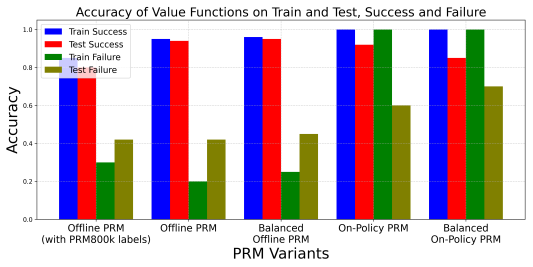


Figure 4: **Outcome Accuracy of PRM**. We evaluate the outcome accuracy of different variants of the process supervision models. Here, both **on-policy** sampling and **label-balancing** leads to higher accuracies for successful and failure trajectories.

Formally, sequential revisions for test-time inference can be viewed as a multi-step and multi-turn Markov Decision Process (MDP). We define a multi-turn revision trajectory τ as $\tau = \{s, a, r, t\}_0^H$ of a bounded horizon H, where each of the N revisions has a sub-horizon (i.e., token lengths) $H_1, H_2, \cdots H_N$. These revision trajectories are constructed from several individual reasoning chains consisting of a prompt x and response y. Here, the response is broken into several semantic steps $a_1, a_2, \cdots a_t$, which when concatenated form the original response y.

Given a dataset of revision traces, we can formally we define a modified MDP $\mathcal{M}_{\text{multiturn}}$ from our Multi-Step, Single-Turn MDP $\mathcal{M}$ (Defined in Section 2) as:

$$\mathcal{M}_{\text{multiturn}} = (\mathcal{S}_{\text{multiturn}}, \mathcal{A}, \mathcal{T}, R, \gamma, \rho_0), \tag{10}$$

where the state $s_i \in \mathcal{S}_{\text{multiturn}}$ at step i in revision turn k consists of the current prompt x, the sequence of current turn reasoning steps $a^{(k)}_{0\ldots i-1}$, and previous turns' reasoning steps $a^{(1\ldots k-1)}_{0\ldots H}$, and the turnwise context $c_{0\ldots k-1}$.

Training in-context verifiers. Given the MDP $\mathcal{M}_{\text{multiturn}}$, we can define familiar reinforcement learning objects such as a policy $\pi(a|s)$ and a value function $Q^{\pi}(s, a)$. These objects can be designed to interact at both a stepwise and turnwise level, enabling a comprehensive supervision process that leverages the history of past revisions and turnwise feedback, to be adaptive to prior attempts.

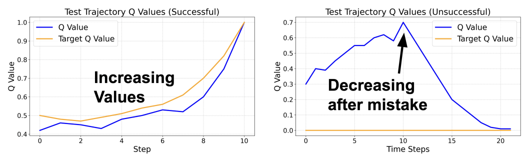


Figure 5: **Learnt PRM Values**: Here we plot the value based PRM for successful and unsuccessful trajectories. For successful trajectories **(left)**, we see monotonically increasing values that match the target Monte-Carlo Return. For negative trajectories **(right)**, we see the value increase and then decrease after a mistake is made, allowing us to identify, where to revise from.

In particular, the value function $Q(s, a)$ can be conditioned on both the current reasoning chain and the steps from prior revisions. Let $\{y_1, y_2, \ldots, y_{k-1}\}$ represent the set of past $k-1$ revisions, where each revision y_j has its own sequence of steps $\{a^{(j)}_1, a^{(j)}_2, \ldots, a^{(j)}_{H_j}\}$. The value function at any step i within the k-th revision can then be expressed as $Q(s_{new}, a)$, where s_{new} is defined as:

$$s_{new} = (x, a^{(1)}_1, a^{(1)}_2, \ldots, a^{(1)}_{H_1}, \ldots, a^{(k)}_i, c_{0\ldots k-1}, k) \tag{11}$$

Here the turnwise context $c_{0\ldots k-1}$ are additional tokens in the form: *"Is the turn correct? [yes/no]"*. This allows the model to understand whether the steps taken in prior revisions were correct or incorrect, providing the value function the outcome of its previous attempts in context.

We can model the cumulative Monte-Carlo return-to-go estimate over all future revisions (given a fixed horizon of revisions), accounting for the potential improvement or deterioration of the solution as further revisions are made. The cumulative return $\mathcal{R}_t$ for step t in the k-th revision is:

$$\mathcal{R}_t = \sum_{i=t}^{H_k} R(s_i, a_i) + \sum_{j=k+1}^{N} \sum_{o=1}^{H_j} R(s_o, a_o) \tag{12}$$

where H_k is the subhorizon of the current revision, N is the total number of revisions, and s_o, a_o denotes the states and actions in future revisions. This cumulative return models the expected future rewards from both the current revision and subsequent revisions, allowing the policy to make decisions that optimize long-term performance across all revision stages.

Intuitively, leveraging the context of prior revisions enables the model to implement effective strategies such as becoming more confident about a certain mistake, allowing for error correction in subsequent attempts, or more confident about previously successful steps, allowing for further positive reinforcement. In contrast, the single-turn value function would potentially lead to the same deterministic set of failure actions over revision turns as the value function is unable to adapt to previous attempts that the policy has tried. This allows the algorithm to be adaptive to new problems, where multiple attempts may be needed to solve a task successfully. Let's now consider additional practical considerations for instantiating such a backtracking framework.

3.3 Practical considerations for Backtracking

Below, we will lay out two practical considerations: (1) tempering distribution with a balanced verifier trained on on-policy samples, and (2) Advantage smoothing for effective backtrack step selection.

Tempering Distribution Shift of PRMs with On-Policy Sampling and Label Balancing. A primary challenge in maintaining the robustness of the value function is the distribution of responses

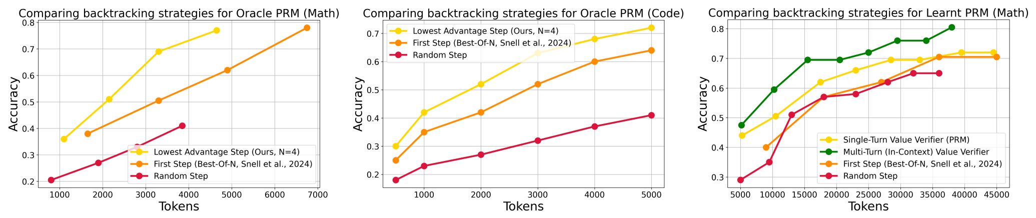


Figure 6: **Backtracking allows for better test time sample efficiency**, measured by generated tokens vs accuracy. Here we allow for up to 4 revisions using the Oracle PRM (math [**left**], code [**middle**]) and 8 revisions with the learnt PRM (**right**). We compare with two baselines: Best-of-N and revising from a random step to show the efficacy of localizing errors with the PRM. The choice of suffix generation after a step to revise from is identified is Best-of-N sampling with N=4 for the oracle PRM and N=16 for the learnt PRM.

it is trained on. Typically, the PRM is initialized using an offline dataset, such as PRM800K, but a significant distribution shift often occurs between the policy's responses during inference and the data used for the initial value function training. This shift can lead to the value function inaccurately assessing the quality of the base policy's step proposals, resulting in sub-optimal outcomes.

To counteract this distribution shift, both the PRM and in-context value verifier are fine-tuned with on-policy samples, following Snell et al. (2024) and Luo et al. (2024), allowing the PRM to better identify specific types of errors made by the proposal model during inference.

Another source of distribution shift is that the set of traces encountered with backtracking at test time are primarily unsuccessful as successful traces are no longer revised as the success threshold would be met. Thus, we introduce a label-balancing mechanism during the fine-tuning of the PRM and in-context value verifier to bring the label distribution closer to what is seen during inference with backtracking. Specifically, let $\mathcal{D} = \mathcal{D}_{\text{positive}} \cup \mathcal{D}_{\text{negative}}$ be a dataset where $\mathcal{D}_{\text{positive}}$ and $\mathcal{D}_{\text{negative}}$ represent successful and unsuccessful trajectories respectively, and both sets are balanced such that $|\mathcal{D}_{\text{positive}}| = |\mathcal{D}_{\text{negative}}|$.

Advantage smoothing for effective backtrack step selection. Depending on the model's capacity and expressivity, value predictions can be incorrect or even ambiguous, leading to the predictions not satisfying basic constraints such as positive monotonicity of values for a trajectory that reaches a desired goal. Therefore, approaches such as temporal smoothing **?** are used in model-based RL to stabilize reward prediction. For backtracking, similar estimation errors in computing the advantage function can lead to suboptimal revision step selections. Thus, we introduce a strategy to smooth the value function during the selection process, which we denote as **minimum advantage with tie margin**, where the revision step with the smallest advantage value is selected, but a margin of tolerance is allowed.

Formally, let $A(i) = A(s_i, a_i)$ denote the advantage for step i in a trajectory $\tau = \{(s_i, a_i)\}_0^H$, and ω denote the tie margin. We identify the minimum advantage value $A_{\min} = \min_i A(i)$, and then select the first step i_{smooth} satisfying:

$$i_{smooth} = \min_j \{ j : |A_{\min} - A(j)| \leq \omega \} \tag{13}$$

This relaxation allows for a more conservative step to be selected for backtracking, reducing likelihood of unrecoverable states, which can significantly hinder the success of the solution chain of thought. In particular, if the revision point precedes where an unrecoverable error is introduced, the revision process can more likely correct the trajectory.

We provide an algorithmic representation for our backtracking framework for multiple turns of sequential revision, given a fixed test-time budget in Algorithm 1.

4 Experimental Evaluation

To evaluate how backtracking would perform on reasoning problems, we consider mathematical reasoning problems from the Hendrycks MATH dataset (Hendrycks et al., 2021) and competitive programming problems from CodeContests (**?**). These datasets span a broad level of mathematical and coding topics ranging from high school to university level. We use the Llama 3.1 8B Instruct Model (Dubey et al., 2024) for MATH and Yi Coder 9B Chat (**?**) for CodeContests. For verification labels and evaluation, we using scoring from Lightman et al. (2023) and **?**. To validate the backtracking framework, we explore three key experimental questions.

1. Can backtracking more efficiently leverage test-time compute than linear search algorithms? Our main results can be found in Figure 6. To measure compute vs performance, we compare the number of generated tokens vs accuracy (coverage). We consider two settings:

1. *Oracle PRM* – In this setting, we use an oracle PRM as our verifier. We compare our advantage-based backtracking algorithm with two baselines, one which backtracks all the way to the first step (i.e., linear search), and one which backtracks to a random step.

2. *Learnt PRM* – For our second setting, we learn two PRMs or value function, one single-turn and other multi-turn (in-context) which conditions on prior-revisions in history. We use the learnt PRM for backtracking and the baselines.

For the oracle PRM, we find that backtracking leads to a $\approx 15\%$ improvement over linear search (backtracking from the first step). We also observe significant performance gain over the random step baseline, demonstrating the necessity of accurate error localization. We see similar trends with the learned value functions, comparing a single-turn value function, multi-turn (in-context) value function, and baselines of revising from the first and random steps.

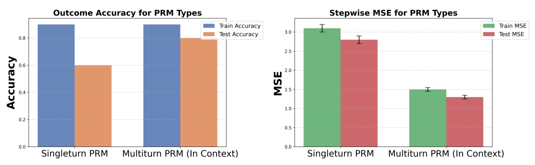


Figure 7: **Outcome and Process Metrics for Value Functions**: We plot the aggregate **(left)** outcome accuracy and **(right)** process-wise MSE from ground truth monte-carlo return-to-go estimate. For both metrics, we find that parameterizing the PRM in the Multi-Turn formulation leads to better performance.

2. How effective are the learnt PRMs at localizing errors for backtracking? To further supplement our findings, we verify in Figure 7 that the learnt PRM indeed scores reasoning steps correctly. We consider two metrics: (1) outcome accuracy of the PRM, allowing us to determine if the PRM can correctly identify if a reasoning chain is incorrect/correct, and (2) step-wise Mean Squared Error (MSE) from the ground truth monte-carlo return-to-go estimate, to evaluate how the PRM performs on intermediate reasoning steps on held-out queries and steps. We additionally use this metric to compare different parameterizations of our value function such as the turn-independent PRM and turn-dependent in-context value verifier. As seen in Figure 7, the absolute performance of the PRM is high for both variants with a high outcome accuracy and low step-wise MSE. Additionally, the in-context, multi-turn PRM can lead to both better outcome accuracy and step-wise MSE. These metrics show the efficacy of the learnt PRM at both an outcome and process level, enabling to use the PRM for both identifying if an error occurred in a reasoning chain and where it did.

Furthermore, we qualitatively examine how the behavior of the PRM (value function) evolves over reasoning steps as seen in Figure 5. For correct solutions, the value function should predict monotonically increasing values, while for incorrect solutions, a drop in the value occurs after a mistake has been made.

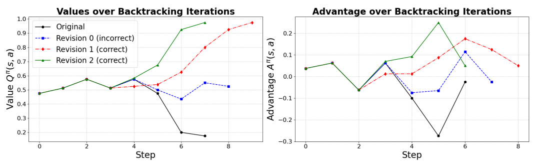


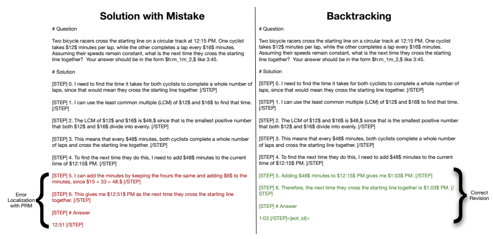
Figure 8: **Evolution of Values over Backtracking Iterations in a Reasoning Problem**: We plot the evolution of **(left)** value (PRM) and **(right)** advantages for steps in a reasoning problem. In revision iterations during backtracking, the value of the trajectory is able to be successfully optimized to be monotonically increasing over steps. The advantage can successfully identify a problematic step midway in the trajectory, leading to successful revisions.

3. Do sequential revisions through backtracking exhibit desired behaviors? We conduct a qualitative analysis of the revision trajectories generated by our backtracking framework. We present a visualization of the revision trajectories in Figure 9, highlighting in an example reasoning problem in the MATH dataset, an error in modular arithmetic that was correctly identified by the PRM.

We also provide visualizations of the evolution of the learnt value function and corresponding advantage over sequential revisions in Figure 8. The advantage is a useful criterion to identify where a mistake has been made in the trajectory, where resampling just a single revision leads to a successful outcome. Additionally, backtracking enables effective ”optimization” over the learnt values, where future revisions have a larger value over reasoning steps than previous revisions.

5 Related Work

Learning and leveraging process-level supervision for LLMs The idea of using process-level supervision was popularized in Uesato et al. (2022) and more recently in Lightman et al. (2023). Both of these works show the promise of using PRMs for MATH. Building upon this idea, papers such as Math-Shepherd (Wang et al., 2023), MiPS (Wang et al., 2024), and OmegaPRM (Luo et al., 2024) present more efficient automated ways to gather data for process-level rewards.



Solution with Mistake

```
# Question

Two bicycle racers cross the starting line on a circular track at 12:15 PM. One cyclist takes $12$ minutes per lap, while the other completes a lap every $16$ minutes. Assuming their speeds remain constant, what is the next time they cross the starting line together?  Your answer should be in the form $h:m_1m_2,$ like 3:45.

# Solution

[STEP] 0. I need to find the time it takes for both cyclists to complete a whole number of laps, since that would mean they cross the starting line together. [/STEP]

[STEP] 1. I can use the least common multiple (LCM) of $12$ and $16$ to find that time. [/STEP]

[STEP] 2. The LCM of $12$ and $16$ is $48,$ since that is the smallest positive number that both $12$ and $16$ divide into evenly. [/STEP]

[STEP] 3. This means that every $48$ minutes, both cyclists complete a whole number of laps and cross the starting line together. [/STEP]

[STEP] 4. To find the next time they do this, I need to add $48$ minutes to the current time of $12:15$ PM. [/STEP]

[STEP] 5. I can add the minutes by keeping the hours the same and adding $6$ to the minutes, since $15 + 33 = 48.$ [/STEP]

[STEP] 6. This gives me $12:51$ PM as the next time they cross the starting line together. [/STEP]

[STEP] # Answer

12:51 [/STEP]
```

Backtracking

```
# Question

Two bicycle racers cross the starting line on a circular track at 12:15 PM. One cyclist takes $12$ minutes per lap, while the other completes a lap every $16$ minutes. Assuming their speeds remain constant, what is the next time they cross the starting line together?  Your answer should be in the form $h:m_1m_2,$ like 3:45.

# Solution

[STEP] 0. I need to find the time it takes for both cyclists to complete a whole number of laps, since that would mean they cross the starting line together. [/STEP]

[STEP] 1. I can use the least common multiple (LCM) of $12$ and $16$ to find that time. [/STEP]

[STEP] 2. The LCM of $12$ and $16$ is $48,$ since that is the smallest positive number that both $12$ and $16$ divide into evenly. [/STEP]

[STEP] 3. This means that every $48$ minutes, both cyclists complete a whole number of laps and cross the starting line together. [/STEP]

[STEP] 4. To find the next time they do this, I need to add $48$ minutes to the current time of $12:15$ PM. [/STEP]

[STEP] 5. Adding $48$ minutes to $12:15$ PM gives me $1:03$ PM. [/STEP]

[STEP] 6. Therefore, the next time they cross the starting line together is $1:03$ PM. [/STEP]

[STEP] # Answer

1:03 [/STEP]<|eot_id|>
```

Figure 9: **Qualitative Example of Revision**: In the example revision, the PRM is able to localize where an error is made in the incorrect solution and correct a modular arithmetic mistake.

Parallel sampling in test-time inference Test-time inference with search has been extensively studied in works such as Feng et al. (2024); Yao et al. (2023); Hao et al. (2023). One key part of this equation is the way samples are selected (Welleck et al., 2024). A common paradigm involves generating multiple trajectories in parallel, then employing some type of model or function to merge these trajectories. In particular, during test-time inference, the integration of a reward model with a proposal distribution (LLM) can be employed to refine the output responses to a given prompt. For instance, search algorithms such as best-of-N (Charniak & Johnson, 2005) and beam search have been explored in works such as Snell et al. (2024), which leverage reward models to select the most promising candidate samples in reasoning tasks. Another notable family of techniques include self-consistency (Wang et al., 2023) and weighted majority voting (Uesato et al., 2022). In self-consistency, the model selects the responses generated with highest frequency across multiple samples. This can be used jointly with our approach, being verifier agnostic.

Iterative revisions in test-time inference An alternative paradigm to parallel sampling in test-time inference is iterative revisions of reasoning steps. Prior work such as RISE (Qu et al., 2024), SCoRe (Kumar et al., 2024) and Self-Refine (Madaan et al., 2023) parameterize a proposal distribution that can correct mistakes in an incorrect solution. This approach is complementary as our fixed proposal distribution can be substituted with the modified learned distribution from these works.

6 Discussion, Conclusion, and Limitations

In this work, we present a framework for sequential response improvement for reasoning problems with PRM-based backtracking. Backtracking allows the model to localize where an error has been made in the response and make targeted revisions to resolve mistakes in a reasoning chain efficiently. We additionally introduce in-context verifiers to allow the verifier to adapt its predictions throughout the search process conditioned on prior attempts at a solution, increasing confidence about a mistake it has made in the past and reinforcing behavior that has led to success. Evaluating this framework with oracle and learned verifiers in the MATH domain, we achieve a $\approx 15\%$ improvement in test-time compute efficiency compared to linear search algorithms.

There are many open questions and limitations. While we used a fixed proposal distribution for the responses, methods like RISE (Qu et al., 2024) use self-improvement to steer the distribution. Could this be combined to enhance sequential corrections? Additionally, our analysis focused on training a verifier on a single reasoning domain. Can we develop a general verifier applicable across domains such as legal reasoning and robotic planning? How does scaling to multiple domains affect performance, especially in areas where stepwise reasoning is less clearly defined, like creative writing?

7 Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

For reproducibility, we provide the following details so readers can replicate the results found in our paper. Firstly, we provide algorithm pseudocode as seen in Algorithm 1, giving the reader transparency in how to replicate the backtracking framework. Additionally, we provide details on how the dataset is curated such as the prompt template as seen in Appendix A.2 and Hyperparameter Details in Appendix A.5. Finally, we provide evaluation details in both the main text in Section 4 and Appendix A.6. For the camera ready, we hope to open-source our on-policy datasets and reward models that we have learned and release a public Github implementation.

REFERENCES

Eugene Charniak and Mark Johnson. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pp. 173–180, USA, 2005. Association for Computational Linguistics. doi: 10.3115/1219840.1219862. URL https://doi.org/10.3115/1219840.1219862.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaoqing Ellen Tan, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aaron Grattafiori, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alex Vaughan, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Franco, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, Danny Wyatt, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Firat Ozgenel, Francesco Caggioni, Francisco Guzmán, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Govind Thattai, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Karthik Prasad, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kun Huang, Kunal Chawla, Kushal Lakhotia, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Maria Tsimpoukelli, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikolay Pavlovich Laptev, Ning Dong, Ning Zhang, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Rohan Maheswari, Russ Howes, Ruty Rinott, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Kohler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vítor Albiero, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaofang Wang, Xiaojian Wu, Xiaolan Wang, Xide Xia, Xilun Wu, Xinbo Gao, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yuchen Hao, Yundi Qian, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, and Zhiwei Zhao. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Xidong Feng, Ziyu Wan, Muning Wen, Stephen Marcus McAleer, Ying Wen, Weinan Zhang, and Jun Wang. Alphazero-like tree-search can guide large language model decoding and training, 2024. URL https://arxiv.org/abs/2309.17179.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model, 2023. URL https://arxiv.org/abs/2305.14992.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset, 2021. URL https://arxiv.org/abs/2103.03874.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei M Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal Behbahani, and Aleksandra Faust. Training language models to self-correct via reinforcement learning, 2024. URL https://arxiv.org/abs/2409.12917.

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 e-prints*, art. arXiv:2305.20050, May 2023. doi: 10.48550/arXiv.2305.20050.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, Jiao Sun, and Abhinav Rastogi. Improve mathematical reasoning in language models by automated process supervision, 2024. URL `https://arxiv.org/abs/2406.06592`.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative Refinement with Self-Feedback. *arXiv e-prints*, art. arXiv:2303.17651, March 2023. doi: 10.48550/arXiv.2303.17651.

Yuxiao Qu, Tianjun Zhang, Naman Garg, and Aviral Kumar. Recursive Introspection: Teaching Language Model Agents How to Self-Improve. *arXiv e-prints*, art. arXiv:2407.18219, July 2024. doi: 10.48550/arXiv.2407.18219.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters. *arXiv e-prints*, art. arXiv:2408.03314, August 2024. doi: 10.48550/arXiv.2408.03314.

Zhiqing Sun, Longhui Yu, Yikang Shen, Weiyang Liu, Yiming Yang, Sean Welleck, and Chuang Gan. Easy-to-Hard Generalization: Scalable Alignment Beyond Human Supervision. *arXiv e-prints*, art. arXiv:2403.09472, March 2024. doi: 10.48550/arXiv.2403.09472.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process- and outcome-based feedback, 2022. URL `https://arxiv.org/abs/2211.14275`.

Peiyi Wang, Lei Li, Zhihong Shao, R. X. Xu, Damai Dai, Yifei Li, Deli Chen, Y. Wu, and Zhifang Sui. Math-Shepherd: Verify and Reinforce LLMs Step-by-step without Human Annotations. *arXiv e-prints*, art. arXiv:2312.08935, December 2023. doi: 10.48550/arXiv.2312.08935.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2023. URL `https://arxiv.org/abs/2203.11171`.

Zihan Wang, Yunxuan Li, Yuexin Wu, Liangchen Luo, Le Hou, Hongkun Yu, and Jingbo Shang. Multi-step problem solving through a verifier: An empirical analysis on model-induced process supervision, 2024. URL `https://arxiv.org/abs/2402.02658`.

Sean Welleck, Amanda Bertsch, Matthew Finlayson, Hailey Schoelkopf, Alex Xie, Graham Neubig, Ilia Kulikov, and Zaid Harchaoui. From decoding to meta-generation: Inference-time algorithms for large language models, 2024. URL `https://arxiv.org/abs/2406.16838`.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023. URL `https://arxiv.org/abs/2305.10601`.

A Appendix

A.1 Algorithm

Algorithm 1 Iterative Backtracking and Solution Revision

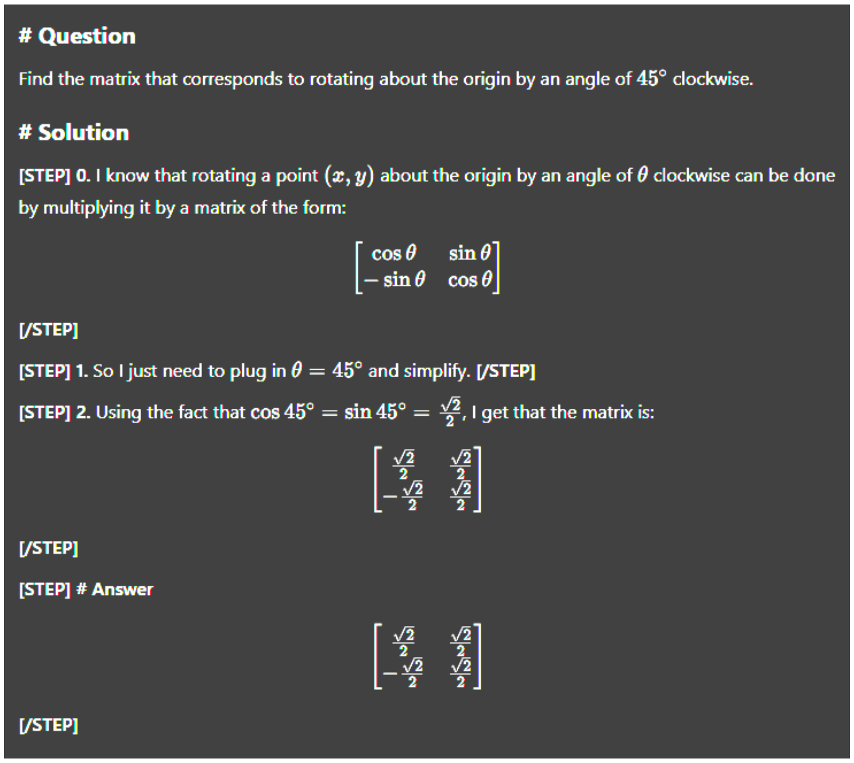
Initial solution trajectory $\tau = \{s_1, s_2, \ldots, s_N\}$, Maximum iterations M, Advantage threshold ω, Success threshold p_{des} Revised solution trajectory τ' $m \leftarrow 0$ $\tau_{\text{best}} \leftarrow \tau$ **Compute** the initial advantage function $A(s_i, a_i)$ for each step i in τ $m < M$ $i_{\text{revise}} \leftarrow \arg\min_i A(s_i, a_i)$ **Compute** i_{smooth} via: $i_{\text{smooth}} = \arg\min_j \{j : |A(s_{i_{\text{revise}}}, a_{i_{\text{revise}}}) - A(s_j, a_j)| \leq \omega\}$ **Resample** the suffix of the trajectory starting from step $T \leftarrow i_{\text{smooth}} - 1$ **Update** the trajectory: $\tau' \leftarrow \{s_1, \ldots, s_{T-1}, s'_T, \ldots, s'_N\}$ **Use** a linear search algorithm (e.g., Best-Of-N, Beam-Search) as a subroutine, conditioning on the prefix up to step $T - 1$ Improved solution found $\tau_{\text{best}} \leftarrow \tau'$ Let (s_N, a_N) be the last state and action in τ_{best} $Q(s_N, a_N) \geq p_{\text{des}}$ **break** Stop if success threshold met. $m \leftarrow m + 1$ **return** τ_{best}

A.2 Dataset Curation

We provide additional details in the training datasets used to train the base policy π_{base} and PRM.

A.2.1 Prompt Template for MATH

We use the prompt template in Figure 10 in the training of our PRM and base policy π_{base} (proposal distribution). We add four additional tokens '[STEP]', '[/STEP]', '[TURN]', '[/TURN]' to the vocabulary of our tokenizer that corresponds to the beginning and end of a step or the beginning and end of a revision.



# Question

Find the matrix that corresponds to rotating about the origin by an angle of 45° clockwise.

# Solution

[STEP] 0. I know that rotating a point (x, y) about the origin by an angle of θ clockwise can be done by multiplying it by a matrix of the form:

$$\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

[/STEP]

[STEP] 1. So I just need to plug in $\theta = 45^\circ$ and simplify. **[/STEP]**

[STEP] 2. Using the fact that $\cos 45^\circ = \sin 45^\circ = \frac{\sqrt{2}}{2}$, I get that the matrix is:

$$\begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix}$$

[/STEP]

[STEP] # Answer

$$\begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix}$$

[/STEP]

Figure 10: **Prompt Template for MATH**: The prompt template above is used for the MATH dataset. Each step and revision turn are surrounded by special start and end tokens.

A.3 Offline Datasets

For the base policy π_{base} and the offline verifier, we utilize the dataloaders from Sun et al. (2024), which study the PRM800K (Lightman et al., 2023) dataset. Here we use all levels of math problems (1-5) for both our policy and PRM datasets. We construct a Monte Carlo Estimate using the ground truth outcome supervision provided in PRM800K (from Stage 1 + 2).

A.4 On-Policy Dataset Collection

The on-policy dataset was collected using the following approach. For each question, four rollouts were generated with the base policy π_{base} (proposal distribution). These rollouts were then decomposed into partial completions, and 20% of these partial completions were further completed using the current policy and evaluated based on the ground truth reward. In the multiturn setup, each question underwent up to a fixed number of revisions, ranging from 0 to 4. To construct multiturn trajectories, with K revisions and N responses per revision, N perm K potential revision trajectories were considered. Given the large number of possible trajectories, the process was simplified by subsampling $J = 100$ trajectories from the set of $\binom{N}{K}$ combinations to avoid redundancy and manage computational complexity.

A.5 Hyperparameters for Value Function Training + Policy Learning

The base policy π_{base} is initialized with SFT using the following hyperparameters:

Name	**Values**
Learning Rate (lr)	1×10^{-6}, 1×10^{-7}
Schedule	Cosine
Warmup Ratio	10%
Model	LLama 3.1 8B Instruct (Dubey et al., 2024), Yi 9B Coder Chat (**?**)

Table 1: Hyperparameters used for SFT

The PRM $Q^{\pi}(s, a)$ is initialized with Monte-Carlo Regression using the following hyperparameters:

Name	**Values**
Learning Rate (lr)	1×10^{-5}, 1×10^{-6}
Schedule	Cosine
Warmup Ratio	10%
PRM Type	Single-Turn, Multi-Turn
Model	LLama 3.1 8B Instruct (Dubey et al., 2024), Yi 9B Coder Chat (**?**)
Discount γ	0.8, 0.9, 1.0

Table 2: Hyperparameters used for PRM Training

A.6 Additional Evaluation Details

We leverage 100 validation queries from the Hendryks Math (Hendrycks et al., 2021) and failure on-policy reasoning chains to construct the dataset for the Performance-Efficiency tradeoff analysis and the evaluation of the learnt PRMs. We ensure that each of the validation queries is unique.