Optimizing Test-Time Compute via Meta Reinforcement Finetuning

Anonymous authors
Paper under double-blind review

Abstract

Training models to efficiently use test-time compute is crucial for improving the reasoning performance of LLMs. While current methods mostly do so via fine-tuning on search traces or running RL against the 0/1 outcome reward, do these approaches efficiently utilize test-time compute? Would these approaches continue to scale as the budget improves? In this paper, we try to answer these questions. We formalize the problem of optimizing test-time compute as a meta reinforcement learning (RL) problem, which provides a principled perspective on spending test-time compute from the lens of exploration and exploitation. It also motivates the use of cumulative regret to measure the efficacy of test-time compute by viewing a long output stream as consisting of several episodes from the model. While current state-of-the-art models do not optimize regret, we show that regret can be minimized by running final 0/1 reward RL regularized by a dense reward bonus, given by the *"information gain"* from each subsequent block in the output stream. We prescribe an approach for quantifying information gain, which measures the utility of an intermediate segment of tokens towards improving accuracy of the final answer. We instantiate this idea to develop ***MRT***, a new class of finetuning methods for optimizing test-time compute. Fine-tuning with ***MRT*** leads to substantial improvements in both performance and token efficiency on the AIME dataset.

1 Introduction

Recent results in LLM reasoning (Snell et al., 2024) illustrate the potential of improving reasoning capabilities by scaling test-time compute. Generally, these approaches train models to produce traces that are longer than the typical correct solution, and consist of tokens that attempt to implement some "algorithm": for example, reflecting on the previous answer (Qu et al., 2024; Kumar et al., 2024), planning (DeepSeek-AI et al., 2025), or implementing some form of linearized search (Gandhi et al., 2024). These approaches explicitly finetune pre-trained LLMs for algorithmic behavior, *e.g.*, SFT on search data (Gandhi et al., 2024; Nie et al., 2024), or the more recent but proprietary outcome reward RL (DeepSeek-AI et al., 2025) methods.

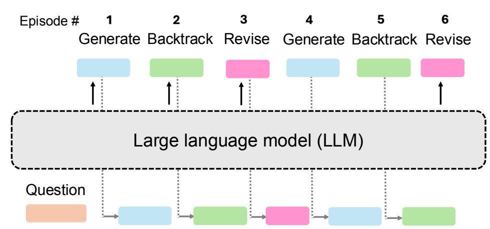

Figure 1: ***Meta reinforcement finetuning (MRT)*** trains an LLM to produce a stream of episodes, which are correlated attempts at solving the input problem (an example is shown above). ***MRT*** poses the problem of optimizing test-time compute as meta RL, which teaches it to optimize cumulative regret over episodes.

Training models to spend test compute by generating long reasoning chains supervised by outcome-reward RL has shown promise. However, to improve continued gains from scaling test-time compute, we will ultimately need to answer the following questions. First, ***are current LLMs efficiently using test-time compute, especially at large test-time budgets?*** That is, do the lengthy thought chains they produce actually help "discover" the correct solution? Second, ***would models be able to improve performance if run at much larger test-time budgets without being trained to do so***? Ultimately, we would want models to derive enough utility from every token (or any semantically meaningful segment) they produce, not only for efficiency but also because doing so imbues a systematic algorithmic procedure to discover solutions for harder, out-of-distribution problems.

In this paper, we formalize the problem of learning to use test-time compute as a meta reinforcement learning (RL) problem (Weng, 2019). Given a query, the LLM generates a stream of output tokens that are segmented into *episodes* (Figure 1). The LLM is trained to maximize test accuracy given a large test-time compute budget, but the exact budget is not known to it. If done right, this should enable generalization to arbitrarily high test-time budgets, beyond what was used for training. An optimal *"budget-agnostic"* LLM should strike a balance between committing to an approach prematurely (*i.e.*, an "exploitation" episode) and attempting high-risk strategies to tackle the problem (*i.e.*, an "exploration" episode), in case one of them succeeds or at least provides information about what not to do. We formalize this notion of "optimality" as minimizing ***cumulative regret*** over the output episodes. Cumulative regret serves two key roles: **(1)** it provides a metric to evaluate how effective SOTA reasoning models such as Deepseek-R1 (DeepSeek-AI et al., 2025) are at effectively utilizing test-time compute; and **(2)** it lends a new class of fine-tuning methods for optimizing test-time compute, that we refer to as ***M****eta* ***R****einforcement fine****T****uning* ***(MRT)***.

Along axis **(1)**, we show that LLMs finetuned with outcome reward RL fail to make steady progress towards discovering the right approach with more episodes. In fact, a much more naïve approach of running substantially fewer episodes coupled with majority voting outperforms outcome-reward RL in terms of both performance and efficiency (Figure 2). Inspired by the lack of progress–which seems critical for discovery of solutions to hard unseen problems–in current models, along axis **(2)**, we utilize the notion of regret to design a family of finetuning approaches that we call ***MRT***. The key idea in ***MRT*** is to provide a reward bonus to each induced episode in the output stream, and optimize it jointly with outcome reward. This reward bonus measures some notion of "progress" made by each episode or alternatively, the amount of ***information*** "gained" by the model through that episode. We instantiate ***MRT*** to train models how to implement backtracking-based search (Figure 10), where we'd expect efficiently utilizing information from prior episodes to be especially helpful in discovering the final answer. ***MRT***-B ("B" to indicate backtrack) extends iterated SFT and RL methods with an information gain based reward bonus.

We evaluate ***MRT***-B on math reasoning tasks. We use ***MRT***-B to finetune Llama3.1-8B and Llama3.2-3B base models on math reasoning problems from the NuminaMATH dataset (Yu et al., 2024). We find that ***MRT***-B outperforms both standard STaR (Zelikman et al., 2022) and RL without any backtracking, as well as methods that spend test-time compute on backtracking and search: outcome-reward RL/STaR (DeepSeek-AI et al., 2025) and self-correction (Qu et al., 2024) approaches. On a token-matched evaluation on AIME, we find that ***MRT***-B attains an improvement of **30%** and **38%** by extending iterated STaR and GRPO respectively. When evaluated primarily on hard, out-of-distribution problems, ***MRT***-B boosts performance by **10%** compared to the best prior approach on this dataset.

2 Preliminaries and Background

Problem setup. Our goal is to optimize LLMs to effectively use test-time compute to tackle difficult problems. We assume access to a reward function $r(\mathbf{x}, \cdot) : \mathcal{Z} \mapsto \{0, 1\}$ that we can query on any output stream of tokens $\mathbf{z}$. For e.g., on a math problem $\mathbf{x}$, with token output stream $\mathbf{z}$, reward $r(\mathbf{x}, \mathbf{z})$ can be one that checks if the $\mathbf{z}$ is correct. We are given a training dataset $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i^*)\}_{i=1}^N$ of problems $\mathbf{x}_i$ and oracle solution traces $\mathbf{y}_i^*$ for each problem that ends at the right answer. Our goal is to use this dataset to train an LLM, which we model as a policy from RL, $\pi(\cdot|\mathbf{x})$. We want to train LLM π to produce a stream of tokens $\mathbf{z}$ on a test problem $\mathbf{x} \sim \mathcal{P}_{\text{text}}$ that achieves a large $r(\mathbf{x}, \mathbf{z})$. We also want the total test-time compute, as measured by the number of tokens in $\mathbf{z}$, to be bounded, but the model is not provided with its value for training to simulate different deployment budgets.

Meta RL primer. RL trains a policy to maximize the reward function. In contrast, the meta RL problem setting assumes access to a distribution of tasks with different reward functions and dynamics. The goal in meta RL is to train a policy on tasks from the training distribution, such that it can do well on the test task. We do not evaluate this policy in terms of its zero-shot performance on the test task, but let it adapt by executing "adaptation" episodes at test time first. Most meta RL methods differ in the design of this adaptation procedure (e.g., in-context RL such as RL^2 (Duan et al., 2016), explicit gradient-based training (Finn et al., 2017b), and latent variable inference (Rakelly et al., 2019)).

3 Problem Formalization: Optimizing Test-Time Compute as Meta RL

In this section, we will formalize the problem of optimizing test-time compute. Then we will show how this formulation can naturally be viewed as a meta RL problem. In the next section, we will show that this meta RL perspective can be used to evaluate if state-of-the-art models are effectively and efficiently using test-time compute (e.g., Deepseek-R1 (DeepSeek-AI et al., 2025)). Finally, we will utilize these ideas to develop a finetuning paradigm, called ***MRT***, to train models to optimize test-time compute.

3.1 Optimizing Test-Time Compute

We want an LLM to attain maximum performance on $\mathcal{D}_{\text{test}}$ within test-time budget C_0.

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

While this is identical to the standard test performance, we emphasize that the budget C_0 used for evaluation is larger than the typical length of a correct response. This means that the LLM $\pi(\cdot|\mathbf{x})$ can afford to spend a part of the token budget into performing operations that do not actually solve $\mathbf{x}$ but rather ***indirectly*** *help* the model in discovering the correct answer eventually. For example, consider a math proof question where the output is composed of a sequence of steps. If the policy could "on-the-fly" determine that it should backtrack a few steps and restart its attempt, it may not only increase its chances of success, but also gain *information* about what steps or strategies to avoid, and which steps to be particularly careful about. However, the LLM π is not aware of the compute budget C_0 before hand during training, and must learn a *"budget-agnostic"* strategy that can work well for all large enough budgets. Therefore, to attain a high test performance, an LLM π should exhibit *adaptive* behavior, that can make the most use of the compute budget available. Doing so requires a different loss than standard outcome-reward RL or STaR, which incentivize the model to learn to directly answer problems, or work well for a specific budget only.

3.2 Characterizing Optimal Use of Test-Time Compute

To develop a training paradigm for using test-time compute effectively, we first need to understand characteristics of *budget-agnostic* LLMs that most optimally use test-time compute. One way to characterize these LLMs is by explicitly segmenting the output stream $\mathbf{z} \sim \pi(\cdot|\mathbf{x})$ into a sequence of *episodes*, and viewing this sequence of episodes as some sort of an "adaptation" procedure on the test problem. Formally, suppose that $\mathbf{z}$ can be divided into k contiguous segments $\mathbf{z} \stackrel{\text{def}}{=} [\mathbf{z}_0, \mathbf{z}_1, \cdots, \mathbf{z}_{k-1}]$[1]. We say an LLM exhibits adaptive behavior if it can use previous episodes $\mathbf{z}_{0:i-1}$ to update its approach in $\mathbf{z}_i$. As shown in Figure 1, these episodes could consist of multiple attempts at a problem (Qu et al., 2024), alternate between verification and generation (Zhang et al., 2024) such that successive generation episodes attain better performance, or be paths in a search tree separated by backtrack markers.

Of course, we eventually want the LLM π to succeed in the last episode it produces within the budget $\mathbf{z}_{k-1}$. However, since the LLM is budget-agnostic, we need to make sure that the LLM is constantly making *progress* and is able to effectively strike the balance between "exploration": producing tokens that are irrelevant to the final answer, but *might* help in later episodes, and "exploitation": attempting to never try out high-risk behaviors.

Building on this intuition, our key insight is that the adaptation procedure implemented in the test-time tokens stream can be viewed as running an RL algorithm on the test problem, where prior episodes serve the role of "training" data for this purely in-context process. Under this abstraction, an "optimal" algorithm is one that makes steady progress towards discovering the solution for the problem with each episode, balancing between discovery and exploitation. As a result, we can use the metric of ***cumulative regret*** from RL to also quantify the optimality of this process.

[1]While there are many different strategies to segment $\mathbf{z}$ into variable number of episodes, for simplicity we assume a fixed number of episodes k in our exposition. Note that if a particular $\mathbf{z}$ contains $l \geq k$ natural episodes, we can always choose to merge the last $l - k$ episodes into one for the purposes of our discussion.

Definition 3.1 (Cumulative regret)**.** Given k episodes $\mathbf{z}$ generated from LLM $\pi(\cdot|\mathbf{x})$, and another LLM μ that computes an estimate (best guess) of the correct response given some episodes, we define cumulative regret as:

$$\Delta_k^{\mu}(\mathbf{x};\pi) \overset{\text{def}}{=} \mathbb{E}_{\mathbf{z}\sim\pi}\left[\sum_{j=0}^{k-1} J_r(\mathbf{x};\pi^*) - J_r(\mu(\cdot|\mathbf{x},\mathbf{z}_{0:j}))\right].$$

Here J_r denotes the expected 0/1 correctness reward attained by LLM μ when conditioning on prior episodes $\mathbf{z}_{0:j-1}$ produced by π and $J_r(\pi^*)$ denotes the reward attained by the best possible budget-agnostic comparator π^* that attains the highest test reward and can be realized by finetuning the base model π_b. The policy μ, that we call the ***meta-prover policy***, could be identical to or different from π itself. For example, if each episode produced by π ends in an estimate of the answer, then we can measure 0/1 correctness of this answer in itself for computing $\tilde{\Delta}_k^{\mu}$ and set $\mu = \pi$. On the other hand, if some episodes produced by π do not end in a final answer that can be scored (e.g., episodes within the "think" block), but do provide helpful *information*, we can use a different μ to help us extrapolate the answer from our existing trajectory. As we will see empirically, μ can be obtained by steering the same LLM π with a system instruction to estimate the best answer so far.

If the regret is large and grows linearly (or faster) as the number of episodes k increases, then we say that episodes $\mathbf{z}$ did not actually make meaningful progress. On the other hand, a low cumulative regret that only grows sublinearly in k indicates that the budget-agnostic LLM π will continue to improve performance as the test-time budget grows.

4 Case Study: Analyzing SoTA DeepSeek-R1

Having defined the notion of cumulative regret, we now use it to analyze if state-of-the-art models, such as DeepSeek-R1 DeepSeek-AI et al. (2025) enjoy a low regret over the episodes they produce. To this end, we compare performance conditioned on different numbers of episodes in the thought block when asking the DeepSeek-R1-Distill-Qwen-32B model to solve problems from two datasets: AIME 2024 and a subset from OmniMATH (Gao et al., 2024). In this context, an *episode* is defined as a continuous segment of thinking uninterrupted by words such as "Wait" and "Alternatively" which break the current flow of logic. More concretely, we compute $[\mathbf{maj@p}]_\mathrm{j}$, in which we truncate the thought block produced by the R1 model to the first j episodes ($\mathbf{z}_{0:j-1}$) and steer it into producing immediate answers (without producing more episodes) conditioned on this truncated thought block. We sample such immediate answers p times and run majority voting over them to produce a single answer. This tells us if adding more episodes in the thought block makes meaningful progress and helps the model to discover the correct solution. We further compare this against the **direct** baseline, in which we finetune the base model to produce "best guess" responses in one episode (we chose the Qwen2.5-32B-Instruct model here since it was trained using the same base model as DeepSeek-R1-Distill-Qwen-32B) (see Appendix D for more details).

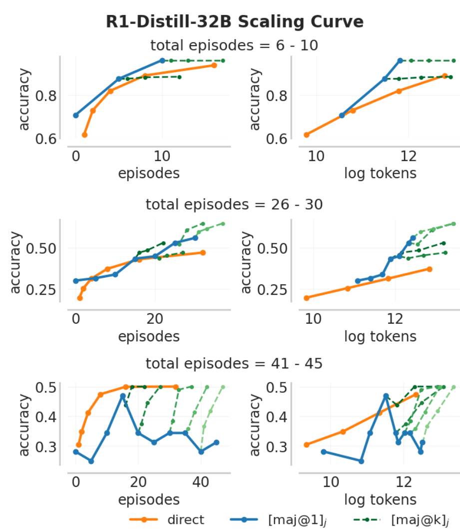

Figure 2: ***R1 scaling curve on OmniMATH subset across different episodes.*** We compare scaling up R1 compute with **direct** pass@k for k = 1, ..., 32 against $[\mathbf{maj@p}]_j$ for p = 1, 2, 8.

Analysis results. We plot the model's average accuracy at different episodes $j = \{0, \cdots, k-1\}$ as a function of the test-time compute (measured in tokens) and the episode index j in Figure 2. To avoid bias induced by averaging across episode lengths, we group solutions based on episode lengths (total episodes = 10, 30, 45). We plot the performance of the direct baseline in orange, and the performance of $[\mathbf{maj@1}]_\mathrm{j}$ at different j in blue. The dashed green lines branching from the blue curve extend each $[\mathbf{maj@1}]_\mathrm{j}$ to $[\mathbf{maj@p}]_\mathrm{j}$ at different $\mathbf{p}$.

Takeaways and implications. When only provided with a few episodes (≤ 10), cumulative regret is low and sequential episodes continuously reduce regret, whereas $[\mathbf{maj@p}]_\mathrm{j}$ and the direct baseline both plateau. However, in "harder" settings that require more episodes (e.g., 41-45; Appendix D), we find that the accuracy of R1 does not increase with each episode and sometimes degrades continuously as more episodes are generated in the output stream. This illustrates that outcome reward RL does not optimize regret, which is significantly high in this case (Figure 2). In addition, simply running the $[\mathbf{maj@p}]_\mathrm{j}$ baseline from intermediate episodes $\mathbf{z}_j$ often results in a higher peak performance under equal compute budgets. For total episodes $\in [41, 45]$, $[\mathbf{maj@p}]_\mathrm{j}$ (green dashed lines) no longer pleateaus and outperforms producing more episodes.

This result is surprising because a long trace with multiple episodes should be capable of implementing the $[\mathbf{maj@p}]_\mathrm{j}$ baseline since there is no new knowledge beyond what the LLM already knows. Inconsistent progress with many episodes likely indicates poor performance as we scale up test-time compute even further. If outcome reward RL was imbuing the LLM with algorithmic procedures, the LLM should be able to improve consistently.

5 MRT: Meta Reinforcement Finetuning

We now turn to developing a finetuning paradigm that we call **meta reinforcement finetuning (MRT)**, which finetunes budget-agnostic LLMs to directly optimize cumulative regret. To build our paradigm, we start by conceptually understanding the failure mode of outcome-reward RL that was used to train R1 which failed to show a consistent reduction in regret with more episodes.

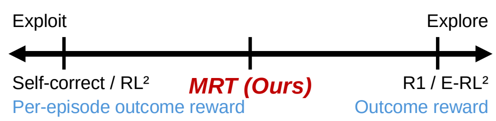

Figure 3: ***Explore/exploit spectrum.*** Final reward RL does not reward intermediate episodes encouraging unstructured exploration, whereas SCoRe (Kumar et al., 2024; Qu et al., 2024) constrains each episode based on its outcome reward making it too exploitative. ***MRT*** strikes a balance by assigning an information gain based reward which aims to make progress in a budget-agnostic setting.

In principle, optimizing the outcome reward over a long stream does not incentivize meaningful progress. As long as the LLM finds *some* arbitrary way to succeed eventually, all intermediate episodes in this rollout will be equally reinforced without accounting for the contribution of every episode towards the eventual success. This is problematic for two reasons: **(i)** in a budget-agnostic setting, we may simply run out of token budget to discover solutions to hard problems if we are not making progress, and **(ii)** we will waste the token budget on easy problems that could be solved otherwise more efficiently. One way of addressing these issues is to directly optimize for the cumulative regret objective (Definition 3.1). We argue that optimizing cumulative regret over budget-agnostic policies is equivalent to optimizing a notion of ***"information gain"*** per episode, which strikes a balance between spending tokens on exploration and exploitation (Figure 3). As we will see in the next section, an episode results in information gain if it reduces uncertainty over a correct answer.

5.1 Gaining Information by Minimizing Regret

In a budget-agnostic setting, one can hope to achieve sub-linear regret only when each episode makes consistent progress. In RL, this notion of progress typically translates to a value function that computes the information or advantage of an action (episode) by marginalizing over all possible future actions (episodes). While appealing, doing so requires aggressive Monte-Carlo sampling and hence, we substitute the value function with J_r from a separate *meta-prover* LLM policy μ, and define information gain from an episode $\mathbf{z}_i$ as the change in average reward obtained by this meta-prover policy μ with and without this episode.

Definition 5.1 (Information gain). Given an episode $\mathbf{z}_j$ from LLM $\pi(\cdot|\mathbf{x})$ and prior context $\mathbf{c}$, and another LLM μ that computes an estimate of the correct response, we define information gain offered by $\mathbf{z}_j$ as

$$I^\mu(\mathbf{z}_j; \mathbf{c}) = J_r(\mu(\cdot|\mathbf{z}_j, \mathbf{c})) - J_r(\mu(\cdot|\mathbf{c})).$$

We will now use this notion of information gain as a dense reward bonus for each intermediate episode, extending ideas from process supervision (Setlur et al., 2024).

5.2 Incorporating Information Gain as a Dense Reward

Defining the standard finetuning loss function based on the expected final reward attained by the last episode as the following objective, ℓ_{FT}:

$$\ell_{\mathrm{FT}}(\pi) := \mathbb{E}_{\mathbf{x}\sim\mathcal{D},\mathbf{z}\sim\pi}\left[r(\mathbf{x}, \mathbf{z}_{k-1})\right], \tag{1}$$

we can train train the LLM π either with the policy gradient obtained by differentiating Equation 1 or with SFT on self-generated data (Singh et al., 2023). We can extend Equation 1 to incorporate the information gain, giving rise to the abstract training objective ($\mathbf{c}$ is the context so far):

$$\ell_{\mathrm{MRT}}(\pi) := \ell_{\mathrm{FT}}(\pi) + \alpha \sum_{j=0}^{k-1} \mathbb{E}_{\substack{\mathbf{c}_k\sim\pi', \\ \mathbf{z}_j\sim\pi(\cdot|\mathbf{c}_k)}} [I^{\mu}(\mathbf{z}_j; \mathbf{c}_k)]. \tag{2}$$

The term in red corresponds to the reward bonus, and it is provided under the distribution of contexts $\mathbf{c}_k$ sampled from prefixes produced by the previous LLM checkpoint, shown as π'. Utilizing the previous policy π' in place of the current policy π serves dual purpose: **(1)** akin to trust-region methods in RL (Schulman et al., 2015; Peng et al., 2019), it allows us to improve over the previous policy provably, and **(2)** it lends ***MRT*** amenable to an offline instantiation based on STaR, which does not run rollouts from the policy after each gradient step. The meta prover policy μ can be any other LLM (e.g., an "-instruct" model which is told to utilize episodes so far to guess the best answer) or the same LLM π itself. We also remark that this additional reward can be provided to the segment of tokens spanning a particular episode ("per-episode" reward) or as a cumulative bonus at the end of the entire test-time trace. Finally, while this objective might appear similar to that of Setlur et al. (2024), we crucially note that the information gain is not an advantage and is computed per episode, and not per step. With this abstract objective in place, we now write down concrete instantiations for SFT and RL.

6 *MRT*-B: Learning to Backtrack

We now instantiate ***MRT*** for training LLMs to implement backtracking-based search in its output stream of tokens. Concretely, we want the LLM to be able to attempt a problem, identify its own mistakes in doing so, and *backtrack* towards a better solution while still repurposing the progress it made in the previous attempts. This problem can benefit from making progress and gaining information from failed episodes to be produce successful episodes eventually.

Formulating backtracking under *MRT*. To instantiate ***MRT*** in this setting, we first need to construct episodes from the output stream of backtracking traces (Figure 5). A convenient way to do so is to split the entire trace at markers that indicate the beginning and end of a backtracking operation. For example, the trace in Figure 10 would be split into three episodes (initial response formatted as a sequence of steps, detecting the error and detecting where to backtrack to, and a revised response). We then optimize the objective in Definition 2 over these episodes with STaR (Zelikman et al., 2022) or RL (Shao et al., 2024). With STaR, this would involve sampling on-policy traces, followed by cloning the ones that succeed not only under the outcome reward, but also attain high information gain. With RL, this would involve adding a reward bonus that corresponds to information gain. We formalize these ideas next.

6.1 STaR and RL Variants of *MRT*-B

The STaR variant of ***MRT***-B uses self-generated rollouts from the base model π_{b} to construct a filtered dataset of traces for SFT. To do so, for each prompt $\mathbf{x}$, we first sample an initial response $\mathbf{z}_0 \sim \pi_{\mathrm{b}}(\cdot|\mathbf{x})$. We then add a backtracking episode $\mathbf{z}_1$, where the model explicitly decides to revert to a previous step in $\mathbf{z}_0$, followed by a new attempt $\mathbf{z}_2$, which shares the prefix with $\mathbf{z}_0$ up until the backtracking step only (see Figure 4). With these traces in hand, we now run filtering to only retain those that **(1)** eventually attain the right answer in $\mathbf{z}_2$, i.e., $r(\mathbf{x}; \mathbf{z}_2) = 1$, and **(2)** have high information backtracks, i.e., $I^{\mu}(\mathbf{z}_{j+1}; \mathbf{x}, \mathbf{z}_{0:j})$ is high, which is computed by rolling out $\mu(\cdot)$. The policy μ is outlined in implementation details. After this, we finetune the LLM, optionally running more than one STaR iteration.

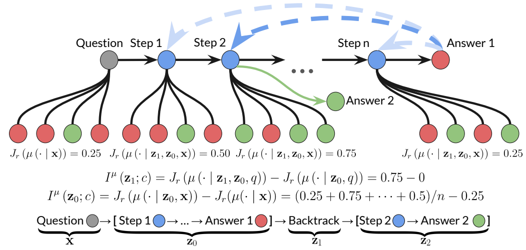

Figure 4: ***On-policy rollout*** generation for ***MRT***-B (STaR). For each context c, ***MRT***-B (STaR) only finetunes on rollouts where $I^{\mu}(\mathbf{z}_i; c) \geq 0$ and which eventually end at the right final answer, i.e., episodes where $r(\mathbf{x}, \mathbf{z}_{k-1}) = 1$.

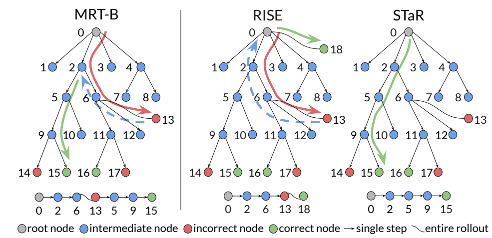

Figure 5: ***Different data construction schemes*** for obtaining warmstart SFT data for learning. ***MRT***-B traverses two paths with the shared prefix, making use of backtracking, which RISE style approaches.

The RL variant of ***MRT***-B implements a similar idea, but using an online RL method (e.g., GRPO (Shao et al., 2024), PPO (Schulman et al., 2017), etc.). The key idea is to now maximize the reward computed in Equation 2 during RL, instead of only the 0/1 outcome reward. There are multiple ways of implementing this procedure including the use of different reward bonuses for different episodes, or just a final reward for all episodes equal to the information-gain adjusted 0/1 outcome reward[2]. We utilize the latter for simplicity and generality of implementation in our experiments.

6.2 Initialization with Warmstart SFT

We found in our preliminary experiments that base pre-trained LLMs (at 3B and 8B parameter ranges) often lacked the ability to sample meaningful backtracking operations due to low coverage over such behavior in the pre-training data. This inability to sample backtracks at all, will severely inhibit learning during RL and STaR. Therefore, before running ***MRT***-B, we had to run an initial phase of "warmstart" supervised finetuning (SFT) to imbue the LLM with a *basis* of backtracking behavior. To do so without human supervision, we generated multiple solution traces by running beam search against the 0/1 outcome reward on every training problem. This beam search did not require any process reward model (PRM) (Snell et al., 2024), since we could simply run additional rollouts to estimate values without any PRM estimation errors. We then generated SFT traces by traversing this tree using a number of heuristics (see Figure 5).

We found that backtracking to nodes in the prefix of an attempt that attain a high estimated success rate, followed by completing the solution from there on resulted in an warmstart SFT dataset that was easy to fit when normalized for the same token budget. On the other hand, SFT datasets generated by stitching arbitrary incorrect solutions from the beam search tree with a correct solution (e.g., RISE) and direct answer traces were both harder to fit as evidenced by the trend in the training loss in Figure 6.

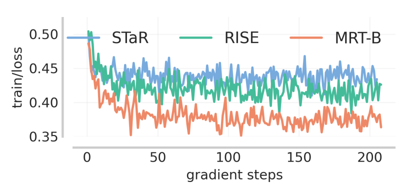

Figure 6: ***Training loss*** for warmstart SFT on multiple data configurations: random stitching ("RISE"), STaR ("rejection sampling"), and our warmstart SFT data ("Backtrack"). A lower loss implies ease of fit.

6.3 Implementation Details

The complete algorithmic implementation for each approach is detailed in Algorithm C.1. The warmstart SFT stage operates on a large dataset of 50k problem-solution pairs sampled from NuminaMath (Li et al., 2024). For each pair, we generated one trace per solution. During each iteration of ***MRT***-B, we use only 20k randomly-sampled problem-solution pairs from NuminaMath. We estimated the information gain reward bonus of backtracking to each of the first 10 steps of the initial episode with 20 rollouts each. The complete set of hyperparameters and training details can be found in Appendix C.2.

[2]Note that both of these implementations will induce the same gradient in expectation, but attain different gradient variance.

7 Experimental Evaluation

The main goal of our experiments is to evaluate the efficacy of ***MRT*** in enabling good use of test-time compute. We situate our experiments in the backtracking problem setting, and run ***MRT***-B. Concretely, we answer the following questions: **(1)** Can ***MRT***-B enable LLMs to successfully discover and backtrack on their mistakes?, **(2)** Does an information gain based reward bonus help the model in discovering nuanced strategies that are better than simply re-attempting the question at hand?, and **(3)** Can the procedure learned by ***MRT***-B generalize to even larger test-time compute budgets than what it was trained on? To this end, we finetune Llama-3.1-8B and Llama-3.2-3B models with STaR and RL variants of ***MRT***-B respectively and perform a number of ablations.

Baselines & experimental setup. We compare ***MRT***-B to existing representative methods: **(a) RISE** (Qu et al., 2024), a self-correction method that teaches LLMs to spend test-time compute on revising their previous responses from the beginning, without backtracking; **(b) STaR** (Zelikman et al., 2022), which improves the LLM's accuracy within one episode using 0/1 rewards and self-training; and **(c) outcome-reward RL w/ GRPO** (Shao et al., 2024), which runs on-policy RL to maximize the 0/1 outcome rewards on top of a backtrack trace. While our training dataset is a subset of NuminaMATH (Li et al., 2024), all responses on these problems are generated from Llama-3.1-8B-Instruct, which also serves as the base model for many of our finetuning runs. We evaluate the performance on a dataset of AIME questions, which is a test bed consisting of unseen, difficult and challenging problems (8B has a pass@10 of $\approx$30% on AIME vs $\approx$60% on our training set).

Evaluation protocol. We evaluate ***MRT***-B in two modes: **(i)** *"parallel"*: for any problem, we sample N independent three-episode traces and compute maj@N and pass@N metrics; and **(ii)** *"linearized"*: following the protocol in Qu et al. (2024), we run N rounds of sequential improvement in a sliding window fashion (i.e. the latest 2048 tokens are retained in context), to still be able to generate long output token streams even when the context length is not that high. We compute pass@N and maj@N evaluations because with models of our size pass@1 numbers on a hard dataset like AIME are expected to only show minor differences.

7.1 Main Performance Results

MRT*-B (STaR).** We start by evaluating the performance of the STaR variant of ***MRT-B with the 8B model. As shown in Figure 7, in both evaluation modes (*parallel* in solid lines; *linearized* in dashed lines), the LLM finetuned by ***MRT***-B attains the highest performance for any given test-time token budget (x-axis) and also the highest peak performance of any method. ***MRT***-B improves token efficiency by over 30% when sampling sequentially. Also observe that running a second iteration of ***MRT***-B, leads to further improvement. Interestingly, while RISE (Qu et al., 2024) based on self-correction without information gain or cumulative regret also improves the final performance compared to the direct STaR approach, it does not do so in a token-efficient manner, falling below the direct approach on the plot.

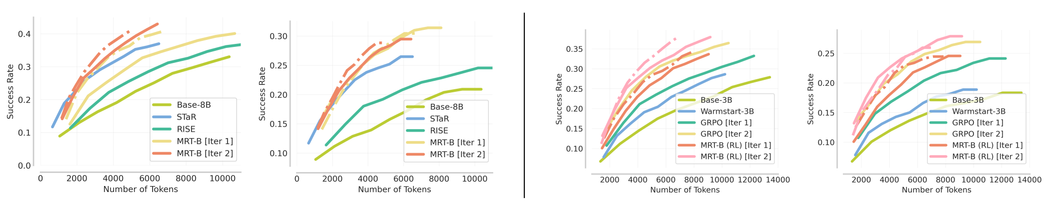

Figure 7: ***MRT*-B (STaR) results (Left)**. We plot pass@10 (left) and maj@10 (right) performance of models on AIME. We also run linearized search (dashed line) for MRT (rest are parallel). For each, we sample k times (1 to 10), and plot pass/maj@k against tokens spent. **MRT-B (RL) results (Right)**. The evaluation protocol is similar to **Left**, where for each method, we sample k times (from 1 to 10), and plot pass/maj@k against tokens spent.

We also note that running ***MRT***-B with linearized evaluations maj@K outperform ***MRT***-B with parallel evaluations maj@K once the test-time compute budget is high enough, indicating that information gain from previous context is useful for attaining higher performance. Finally, also note that running multiple iterations of ***MRT***-B (STaR) improves performance as shown upto 2 iterations in Figure 7.

***MRT*-B (RL).** Next, we test the on-policy version of our approach using the GRPO (Shao et al., 2024) algorithm, finetuning the 3B model with the warmstart SFT as initialization. As discussed in Section 6.3, in this case, the information gain bonus is added to the entire episode. As shown in Figure 7, we find that ***MRT*-B** (RL) with GRPO is able to improve the compute efficiency of linearized maj@K by 38% compared to the strong baseline of GRPO with only the final outcome reward, that has been fundamental to the recent successes of proprietary reasoning models (DeepSeek-AI et al., 2025). Again we note that not only does RL improve token efficiency (i.e., better performance for a given token budget), it also improves the peak performance. In the next section, we show that this is because our approach actually discovers new solutions on the harder questions.

7.2 Ablation Studies and Diagnostic Visualizations

Next, we perform controlled experiments to better understand the reasons behind the efficacy of ***MRT*-B**. First, we aim to understand if the reward bonus introduced by ***MRT*-B** is effective at discovering solutions. To this end, we specifically stress-test ***MRT*-B** and its counterparts trained without information gain (i.e., RISE and naïve outcome-reward RL) on *hard* problems on which the base model generally fails. Hard problems are those on which the warmstart SFT model attains a low performance ($\leq 10\%$). We find that the difference in pass@10 performance between ***MRT*-B** and its counterpart without information gain is even larger in this setting: notably, ***MRT*-B** (RL) solves **60** additional AIME test problems (out of 660 hard questions) that naïve GRPO cannot solve. This substantial gap indicates that incentivizing information gain is crucial for performance on hard questions, where models need to discover new solutions.

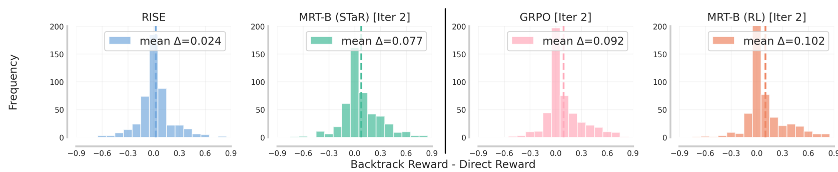

Figure 8: ***Information gain histograms*** over the backtracking episode for RISE and ***MRT*-B** (STaR) on the left and GRPO and ***MRT*-B** (RL) on right, computed on the evaluation set. In each case, using reward values prescribed by ***MRT*-B** amplifies information gain on the test-time trace, enabling it to make consistent progress.

We visualize information gain (Definition 5.1) histograms obtained by running many episodes in the evaluation rollouts produced after finetuning with ***MRT*-B**, and compare it with its counterpart without the bonus (Figure 8). Observe that both ***MRT*-B** (STaR) and ***MRT*-B** (RL) exhibit a net positive and higher information gain over the backtracking episode compared to RISE and outcome-reward RL respectively. This shows that by incorporating the reward bonus, ***MRT*** amplifies information gain at test time as well.

Our final ablation study compares ***MRT*-B** (RL) to a variant of outcome-reward RL that utilizes a length penalty. Concretely, this approach rewards an on-policy rollout with a length-normalized final outcome reward, preferring shorter sequences in order to explicitly improve efficiency of test compute. We find in Figure 9 that this approach is more efficient at small token budgets and produces much shorter responses, but surprisingly it attains a worse peak performance, perhaps because it does not explicitly incentivize the model to make progress and only prefers shorter lengths.

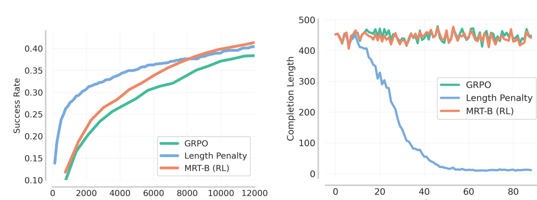

Figure 9: ***Information gain vs. length penalty***. **Left:** pass@k scaling curves on AIME. While length-adjusted outcome reward attains better token efficiency, running ***MRT*-B** (RL) with a reward bonus outperforms it in peak performance and efficiency with more linearized evaluation. **Right:** evolution of length distributions in RL training.

Conclusion. We formalize optimizing test-time compute via meta-RL and introduce cumulative regret as a measure of its efficiency. Our paradigm, ***MRT***, finetunes LLMs to minimize regret by making steady progress via a dense reward bonus. Empirically, ***MRT*** improves performance when provided with more test-time compute. Future work should study longer training contexts, flexible notions of episodes, and scale ***MRT*-B** to use branched rollouts and, hence, exactly optimize the cumulative regret metric.

References

Agarwal, R., Liang, C., Schuurmans, D., and Norouzi, M. Learning to generalize from sparse and underspecified rewards. In *International conference on machine learning*, pp. 130–140. PMLR, 2019.

Beck, J., Vuorio, R., Liu, E. Z., Xiong, Z., Zintgraf, L., Finn, C., and Whiteson, S. A survey of meta-reinforcement learning. *arXiv preprint arXiv:2301.08028*, 2023.

Beeching, E., Tunstall, L., and Rush, S. Scaling test-time compute with open models. URL https://huggingface.co/spaces/HuggingFaceH4/blogpost-scaling-test-time-compute.

Chen, X., Xu, J., Liang, T., He, Z., Pang, J., Yu, D., Song, L., Liu, Q., Zhou, M., Zhang, Z., et al. Do not think that much for 2+ 3=? on the overthinking of o1-like llms. *arXiv preprint arXiv:2412.21187*, 2024.

Chow, Y., Tennenholtz, G., Gur, I., Zhuang, V., Dai, B., Thiagarajan, S., Boutilier, C., Agarwal, R., Kumar, A., and Faust, A. Inference-aware fine-tuning for best-of-n sampling in large language models. *arXiv preprint arXiv:2412.15287*, 2024.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

DeepSeek-AI, Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., Zhang, X., Yu, X., Wu, Y., Wu, Z. F., Gou, Z., Shao, Z., Li, Z., Gao, Z., Liu, A., Xue, B., Wang, B., Wu, B., Feng, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., Dai, D., Chen, D., Ji, D., Li, E., Lin, F., Dai, F., Luo, F., Hao, G., Chen, G., Li, G., Zhang, H., Bao, H., Xu, H., Wang, H., Ding, H., Xin, H., Gao, H., Qu, H., Li, H., Guo, J., Li, J., Wang, J., Chen, J., Yuan, J., Qiu, J., Li, J., Cai, J. L., Ni, J., Liang, J., Chen, J., Dong, K., Hu, K., Gao, K., Guan, K., Huang, K., Yu, K., Wang, L., Zhang, L., Zhao, L., Wang, L., Zhang, L., Xu, L., Xia, L., Zhang, M., Zhang, M., Tang, M., Li, M., Wang, M., Li, M., Tian, N., Huang, P., Zhang, P., Wang, Q., Chen, Q., Du, Q., Ge, R., Zhang, R., Pan, R., Wang, R., Chen, R. J., Jin, R. L., Chen, R., Lu, S., Zhou, S., Chen, S., Ye, S., Wang, S., Yu, S., Zhou, S., Pan, S., Li, S. S., Zhou, S., Wu, S., Ye, S., Yun, T., Pei, T., Sun, T., Wang, T., Zeng, W., Zhao, W., Liu, W., Liang, W., Gao, W., Yu, W., Zhang, W., Xiao, W. L., An, W., Liu, X., Wang, X., Chen, X., Nie, X., Cheng, X., Liu, X., Xie, X., Liu, X., Yang, X., Li, X., Su, X., Lin, X., Li, X. Q., Jin, X., Shen, X., Chen, X., Sun, X., Wang, X., Song, X., Zhou, X., Wang, X., Shan, X., Li, Y. K., Wang, Y. Q., Wei, Y. X., Zhang, Y., Xu, Y., Li, Y., Zhao, Y., Sun, Y., Wang, Y., Yu, Y., Zhang, Y., Shi, Y., Xiong, Y., He, Y., Piao, Y., Wang, Y., Tan, Y., Ma, Y., Liu, Y., Guo, Y., Ou, Y., Wang, Y., Gong, Y., Zou, Y., He, Y., Xiong, Y., Luo, Y., You, Y., Liu, Y., Zhou, Y., Zhu, Y. X., Xu, Y., Huang, Y., Li, Y., Zheng, Y., Zhu, Y., Ma, Y., Tang, Y., Zha, Y., Yan, Y., Ren, Z. Z., Ren, Z., Sha, Z., Fu, Z., Xu, Z., Xie, Z., Zhang, Z., Hao, Z., Ma, Z., Yan, Z., Wu, Z., Gu, Z., Zhu, Z., Liu, Z., Li, Z., Xie, Z., Song, Z., Pan, Z., Huang, Z., Xu, Z., Zhang, Z., and Zhang, Z. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Dorfman, R., Shenfeld, I., and Tamar, A. Offline meta learning of exploration. *arXiv preprint arXiv:2008.02598*, 2020.

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. Rl ˆ2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. *CoRR*, abs/1703.03400, 2017a. URL http://arxiv.org/abs/1703.03400.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pp. 1126–1135. PMLR, 2017b.

Gandhi, K., Lee, D., Grand, G., Liu, M., Cheng, W., Sharma, A., and Goodman, N. D. Stream of search (sos): Learning to search in language. *arXiv preprint arXiv:2404.03683*, 2024.

Gao, B., Song, F., Yang, Z., Cai, Z., Miao, Y., Dong, Q., Li, L., Ma, C., Chen, L., Xu, R., et al. Omni-math: A universal olympiad level mathematic benchmark for large language models. *arXiv preprint arXiv:2410.07985*, 2024.

Ghosh, D., Rahme, J., Kumar, A., Zhang, A., Adams, R. P., and Levine, S. Why generalization in RL is difficult: Epistemic pomdps and implicit partial observability. *CoRR*, abs/2107.06277, 2021. URL https://arxiv.org/abs/2107.06277.

Gupta, A., Eysenbach, B., Finn, C., and Levine, S. Unsupervised meta-learning for reinforcement learning. *CoRR*, abs/1806.04640, 2018a. URL http://arxiv.org/abs/1806.04640.

Gupta, A., Mendonca, R., Liu, Y., Abbeel, P., and Levine, S. Meta-reinforcement learning of structured exploration strategies. *CoRR*, abs/1802.07245, 2018b.

Jones, A. L. Scaling scaling laws with board games. *arXiv preprint arXiv:2104.03113*, 2021.

Kamienny, P.-A., Pirotta, M., Lazaric, A., Lavril, T., Usunier, N., and Denoyer, L. Learning adaptive exploration strategies in dynamic environments through informed policy regularization. *arXiv preprint arXiv:2005.02934*, 2020.

Kang, K., Wallace, E., Tomlin, C., Kumar, A., and Levine, S. Unfamiliar finetuning examples control how language models hallucinate, 2024.

Kimi-Team. Kimi k1.5: Scaling reinforcement learning with llms, 2025.

Kumar, A., Zhuang, V., Agarwal, R., Su, Y., Co-Reyes, J. D., Singh, A., Baumli, K., Iqbal, S., Bishop, C., Roelofs, R., et al. Training language models to self-correct via reinforcement learning. *arXiv preprint arXiv:2409.12917*, 2024.

Lehnert, L., Sukhbaatar, S., Su, D., Zheng, Q., Mcvay, P., Rabbat, M., and Tian, Y. Beyond a*: Better planning with transformers via search dynamics bootstrapping. *arXiv preprint arXiv:2402.14083*, 2024.

Li, J., Beeching, E., Tunstall, L., Lipkin, B., Soletskyi, R., Huang, S., Rasul, K., Yu, L., Jiang, A. Q., Shen, Z., et al. Numinamath: The largest public dataset in ai4maths with 860k pairs of competition math problems and solutions. *Hugging Face repository*, 13:9, 2024.

Liu, E. Z., Raghunathan, A., Liang, P., and Finn, C. Decoupling exploration and exploitation for meta-reinforcement learning without sacrifices. In *International conference on machine learning*, pp. 6925–6935. PMLR, 2021.

Mendonca, R., Gupta, A., Kralev, R., Abbeel, P., Levine, S., and Finn, C. Guided meta-policy search. *Advances in Neural Information Processing Systems*, 32, 2019.

Moon, S., Park, B., and Song, H. O. Guided stream of search: Learning to better search with language models via optimal path guidance. *arXiv preprint arXiv:2410.02992*, 2024.

Nie, A., Su, Y., Chang, B., Lee, J. N., Chi, E. H., Le, Q. V., and Chen, M. Evolve: Evaluating and optimizing llms for exploration. *arXiv preprint arXiv:2410.06238*, 2024.

Peng, X. B., Kumar, A., Zhang, G., and Levine, S. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

Qu, Y., Zhang, T., Garg, N., and Kumar, A. Recursive introspection: Teaching language model agents how to self-improve. *arXiv preprint arXiv:2407.18219*, 2024.

Rakelly, K., Zhou, A., Finn, C., Levine, S., and Quillen, D. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International conference on machine learning*, pp. 5331–5340. PMLR, 2019.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In Bach, F. and Blei, D. (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1889–1897, Lille, France, 07–09 Jul 2015. PMLR.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

Setlur, A., Nagpal, C., Fisch, A., Geng, X., Eisenstein, J., Agarwal, R., Agarwal, A., Berant, J., and Kumar, A. Rewarding progress: Scaling automated process verifiers for llm reasoning. *arXiv preprint arXiv:2410.08146*, 2024.

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y., Wu, Y., et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Singh, A., Co-Reyes, J. D., Agarwal, R., Anand, A., Patil, P., Garcia, X., Liu, P. J., Harrison, J., Lee, J., Xu, K., et al. Beyond human data: Scaling self-training for problem-solving with language models. *arXiv preprint arXiv:2312.06585*, 2023.

Snell, C., Lee, J., Xu, K., and Kumar, A. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Stadie, B. C., Yang, G., Houthooft, R., Chen, X., Duan, Y., Wu, Y., Abbeel, P., and Sutskever, I. Some considerations on learning to explore via meta-reinforcement learning, 2019. URL https://arxiv.org/abs/1803.01118.

Welleck, S., Bertsch, A., Finlayson, M., Schoelkopf, H., Xie, A., Neubig, G., Kulikov, I., and Harchaoui, Z. From decoding to meta-generation: Inference-time algorithms for large language models. *arXiv preprint arXiv:2406.16838*, 2024.

Weng, L. Meta reinforcement learning. *lilianweng.github.io*, 2019. URL https://lilianweng.github.io/posts/2019-06-23-meta-rl/.

Wu, Y., Sun, Z., Li, S., Welleck, S., and Yang, Y. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*, 2024.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., and Narasimhan, K. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023.

Yin, M., Tucker, G., Zhou, M., Levine, S., and Finn, C. Meta-learning without memorization. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=BklEFpEYwS.

Yu, L., Jiang, W., Shi, H., Yu, J., Liu, Z., Zhang, Y., Kwok, J. T., Li, Z., Weller, A., and Liu, W. Metamath: Bootstrap your own mathematical questions for large language models, 2024.

Zelikman, E., Wu, Y., Mu, J., and Goodman, N. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

Zhang, L., Hosseini, A., Bansal, H., Kazemi, M., Kumar, A., and Agarwal, R. Generative verifiers: Reward modeling as next-token prediction. *arXiv preprint arXiv:2408.15240*, 2024.

Appendices

A Related Work

Scaling test-time compute. Recent works Snell et al. (2024); Jones (2021) show that scaling test-time compute can be more optimal than scaling data or model parameters. Early works Wu et al. (2024); Welleck et al. (2024) on test-time compute train separate verifiers Setlur et al. (2024); Chow et al. (2024) for best-of-N Cobbe et al. (2021) or beam search Beeching et al.. Building on this, recent work (Gandhi et al., 2024; Moon et al., 2024) trains LLMs to "simulate" in-context search at test time by training them to imitate manually-stitched search traces. However, gains from such approaches are limited since finetuning on search traces that are unfamiliar to the base model can lead to memorization (Kumar et al., 2024; Kang et al., 2024). As a result, ***MRT***-B designs the data for warmstart SFT carefully and then runs on-policy STaR/RL, which does not suffer from this issue (Kumar et al., 2024; Qu et al., 2024). More recently, RL with outcome rewards has shown promise for finetuning LLMs to produce long CoTs that can search Lehnert et al. (2024), plan Yao et al. (2023), introspect Qu et al. (2024) and correct DeepSeek-AI et al. (2025); Kimi-Team (2025). We show that RL with outcome rewards alone does not learn to discover solutions by making use of test-time compute efficiently, corroborating prior evidence with overly long solutions (Chen et al., 2024). Our approach ***MRT*** poses optimizing test-time compute as a meta RL problem to make good use of test-time compute.

Meta RL. ***MRT*** formulates optimizing test-time compute as a meta RL problem (Beck et al., 2023; Gupta et al., 2018b;a). In fact, prior work in RL (Ghosh et al., 2021; Rakelly et al., 2019) shows that it is *necessary* to solve a meta RL problem to effectively generalize to unseen initial states or contexts (i.e., new test problems in our case), with a little bit of interaction (i.e., initial episodes in the token history). Most prior work in meta RL (Finn et al., 2017a; Agarwal et al., 2019; Mendonca et al., 2019) differs in the design of the adaptation procedure employed. ***MRT*** is closest to meta RL methods that use in-context histories (Duan et al., 2016; Stadie et al., 2019), but differs in the design of the reward bonus, striking a balance between E-RL^2 (Stadie et al., 2019) that does not reward all but the last episode (only exploration), and RL^2 (Duan et al., 2016) that rewards each episode based on its success rate (only exploitation). Other works (Kamienny et al., 2020; Dorfman et al., 2020) try to jointly optimize for exploration and exploitation, but often collapse to simply memorizing the direct solution because of a "chicken-and-egg" problem at the start of learning (Liu et al., 2021; Yin et al., 2020). Our paradigm ***MRT*** alleviates this problem with the use of an information gain bonus under decoupled meta-prover policies μ.

B Example of Backtrack

Example of Backtrack

Query: A type of operation is defined as follows: $\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix} = ad - bc\right)$. The conjugate of the complex number $\left(\begin{bmatrix} 1+i & -1 \\ 2 & 3i \end{bmatrix}\right)$ is ______.

Solution: $\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right)$ is defined as $(ad - bc)$, but this doesn't appear to be a matrix operation. Instead, it seems to be a determinant operation. The problem statement seems to be a bit unclear, but if we consider the given complex number $\left(\begin{bmatrix} 1+i & -1 \\ 2 & 3i \end{bmatrix}\right)$ as a 2×2 matrix, we can calculate its determinant, which would be the conjugate.
Step 1: To find the conjugate of the complex number $\left(\begin{bmatrix} 1+i & -1 \\ 2 & 3i \end{bmatrix}\right)$, we first need to calculate the determinant of this matrix using the formula: $(ad - bc)$.
Step 2: From the given matrix, we have $a = 1 + i$, $b = -1$, $c = 2$, and $d = 3i$.
Step 3: The determinant of the matrix is $((1 + i)(3i) - (-1)(2)) = (3i^2 + 2)$.
Step 4: Simplify the expression by substituting (i^2) with -1: $(3(-1) + 2) = (-3 + 2)$.
Step 5: Calculate the expression: $(-3 + 2 = -1)$.
Step 6: The conjugate of a complex number $(a + bi)$ is $(a - bi)$, so the conjugate of (-1) is (-1).
Answer: $\boxed{-1}$ ✗
Wait, this seems off. Let's try something else.
Step 3: Plugging these values into the determinant formula, we get $(1 + i)(3i) - (-1)(2) = 3i^2 + 3i + 2 = -3 + 3i + 2 = -1 + 3i$.
Step 4: However, the conjugate of a complex number is obtained by changing the sign of the imaginary part. Therefore, the conjugate of $-1+3i$ is $-1-3i$. Step 5: Therefore, the conjugate of the given complex number $\left(\begin{bmatrix} 1+i & -1 \\ 2 & 3i \end{bmatrix}\right)$ is $(-1 - 3i)$).
Answer: $\boxed{-1 - 3i}$ ✓

Figure 10: **Example of backtrack trajectory used to train the model.** The trajectory shows that the model first try to solve the problem, then it recognized that the prior solution is wrong from step 3, therefore, the model backtrack to step 2 in the prior solution and redo step 3 with correction. The mistake is highlighted in red, the correction is highlighted in green, and the backtracking step detection is highlighted in yellow.

C Implementation Details

C.1 Pseudocode

Algorithm 1 ***MRT*-B** (STaR)

Input base model π_{θ_b}; problems $\mathcal{D}$
model $\pi_\theta \leftarrow \pi_{\theta_b}$, fine-tune dataset $\mathcal{D}_{\text{ft}} \leftarrow \emptyset$
for iteration = 1, ..., T **do**
for $q \in \mathcal{D}$ **do**
Sample 1 rollout $z_0 \sim \pi_{\theta_b}(\cdot|q)$.
Compute reward $\{r_0\}$ for z_0 as in Definition 5.1
Sample G outputs $\{z^i_{1:2}\}^G_{i=1} \sim \pi_\theta(\cdot|q, z_0)$
Compute rewards $\{r_i\}^G_{i=1}$ for each sampled output $z^i_{1:2}$ as in Definition 5.1
$\mathcal{D}_{\text{ft}} \leftarrow \mathcal{D}_{\text{ft}} \cup \left\{(q, z_0, z^i_{1:2})|r_i > r_0, i = \arg\max_i\{r_i\}^G_{i=1}\right\}$
end for
$\pi_\theta \leftarrow$ Fine-tune π_{θ_b} with $\mathcal{D}_{\text{ft}}$
end for

Algorithm 2 ***MRT*-B** (RL)

Input base model π_{θ_b}; problems $\mathcal{D}$
model $\pi_\theta \leftarrow \pi_{\theta_b}$
for iteration = 1, ..., T **do**
$\pi_{\text{ref}} \leftarrow \pi_\theta$
for step = 1, ..., k **do**
Sample a batch $\mathcal{D}_b$ from $\mathcal{D}$
Update the old policy model $\pi_{\theta_{old}} \leftarrow \pi_\theta$
for $q \in \mathcal{D}_b$ **do**
Sample 1 rollout $z_0 \sim \pi_{\theta_b}(\cdot|q)$.
Sample G outputs $\{z^i_{1:2}\}^G_{i=1} \sim \pi_{\theta_{old}}(\cdot|q, z_0)$
Compute rewards $\{r_i\}^G_{i=1}$ for each sampled output $z^i_{1:2}$ as in Definition 5.1
Compute $\hat{A}_{i,t}$ for the t-th token of z_i through group relative advantage estimation
end for
for GRPO iteration = 1, ..., μ **do**
Update the policy model π_θ by maximizing the GRPO objective
end for
end for
end for

C.2 Hyperparameters

For ***MRT***-B (STaR), we utilize the trl codebase, but we customize the loss function to be weighted by information gain defined in Definition 5.1. The base models are directly loaded from Hugging Face: Llama-3.1-8B-Instruct. The hyperparameters used for finetuning are specified in Table 2.

Hyperparameter	Values
learning_rate	1.0e-6
num_train_epochs	3
batch_size	256
gradient_checkpointing	True
max_seq_length	4096
bf16	True
num_gpus	8
learning rate	1e-6
warmup ratio	0.1

Table 1: Hyperparameters used for ***MRT***-B (STaR)

For ***MRT***-B (RL), we utilize the open-r1 codebase, but we customize the loss function to be weighted by information gain defined in Definition 5.1. The base models are directly loaded from Hugging Face: meta-llama/Llama-3.2-3B-Instruct. The hyperparameters used for finetuning are specified in Table 2.

Hyperparameter	Values
learning_rate	1.0e-6
lr_scheduler_type	cosine
warmup_ratio	0.1
weight_decay	0.01
num_train_epochs	1
batch_size	256
max_prompt_length	1500
max_completion_length	1024
num_generations	4
use_vllm	True
vllm_gpu_memory_utilization	0.8
temperature	0.9
bf16	True
num_gpus	8
deepspeed_multinode_launcher	standard
zero3_init_flag	true
zero_stage	3

Table 2: Hyperparameters used for ***MRT***-B (RL)

D Full Analysis of DeepSeek-R1

In this section we will give a more detailed outline on how the study on DeepSeek-R1 is done.

We focus our analysis primarily on a subset of 40 problems taken from Omni-MATH. We chose Omni-MATH because it is not an explicit benchmark that DeepSeek-R1 mentioned in their report and therefore could be a better representation of problems encountered outside of their selected benchmarks. We chose these 10 problems from each of the difficulty levels 4, 4.5, 5, and 5.5. The reason for doing this is to avoid the trivial cases of getting an accuracy close to either 0 or 100. To look at how R1 performs on a benchmark it did mention in its report, we additionally show experiments done on 30 problems from AIME 2024.

The first step in generating the data is to obtain the main "stems" which we will later truncate to construct solutions of different episode lengths. To do so, we sample 4 stems by querying the model problems in our datasets at a temperature of 0.7 and 8192 max tokens. The same is done to obtain our pass@k baseline for the direct approach, except that we take k = 32 to simulate pass@32.

After we have obtained our stems, we separate them into episodes by filtering for explicit phrases that indicate a disruption in the natural flow of logic. We further constrain each episode to be at least three steps (each "step" is an entry separated by the delimiter "\n\n") to avoid consecutive trivial episodes. The explicit phrases are listed in Figure 12. If a step begins with one of these phrases, then we consider it to be the beginning of a new episode.

The number of episodes depends on the problem and particular solution that was sampled. The distribution is shown in Figure 15. Due to the large number of episodes, we group the episodes in groups of 5 for Omni-MATH and groups of 3 for AIME, so each dot on the blue curve in Figures 13 and 14 represent 5 or 3 episodes.

For each episode prefix $\mathbf{z}_{0:j-1}$ ($j \equiv 0 \mod 5$ or 3), we ask the model to stop taking further episodes, summarize the existing work, and give an answer (i.e., $\mu(\cdot|\mathbf{x}, \mathbf{z}_{0:j-1})$). To do so, we leverage R1's tendency to give answers after it indicates that it has thought for too long and the ⟨think⟩ tag is closed with ⟨\think⟩. The model is asked to give an answer in this way 8 times to simulate maj@8, at temperature 0.7 and 4096 max tokens. We show the prompt in . With the accuracy of these rollouts we are able to compute the blue and green curves in Figures 13 and 14.

Explicit step prefixes for separating episodes in R1 solution

Wait
But wait
Alternatively
Is there another way to think about this?
But let me double-check
But hold on

Figure 11: **Explicit step prefixes for separating episodes in R1 solution**. This is a list of phrases that indicate a disturbance in the natural flow of logic under R1. If a step begins with one of these phrases, we consider it the start of a new episode.

Prompt used to extract answer from R1

{Insert $\mathbf{x}, \mathbf{z}_{0:j-1}$ here (⟨think⟩ tag will be part of $\mathbf{z}_{0:j-1}$)}

Time is up.

Given the time I've spent and the approaches I've tried, I should stop thinking and formulate a final answer based on what I already have.
⟨\think⟩

Step-by-Step Explanation and Answer:

1. **

Figure 12: **Prompt used to extract answer from R1**. We use the prompt above to simulate $\mu(\cdot|\mathbf{x}, \mathbf{z}_{0:j-1})$ and extract an answer after j episodes.

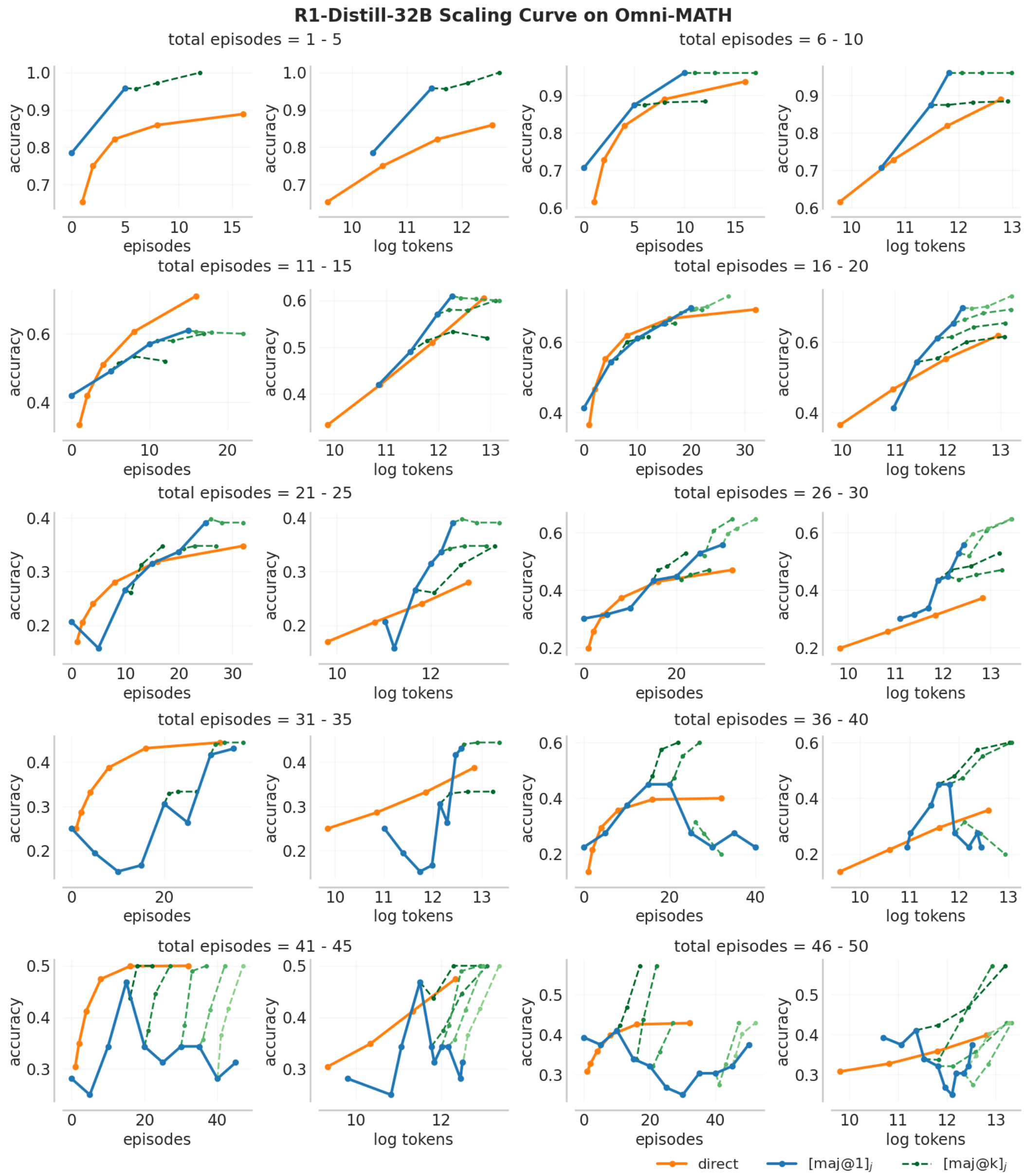

Figure 13: **R1 scaling curve on Omni-MATH subset across different episodes**. We compare scaling up R1 compute with **direct** pass@k for k = 1, 2, 8, 16, 32 against [**maj@p**]$_j$ for p = 1, 2, 8 and varying levels of j. Note that the total episodes matches the length of the blue curve. It is a range rather than a single number due to the concatenation of episodes into groups of 5 and 3 mentioned in the full analysis.

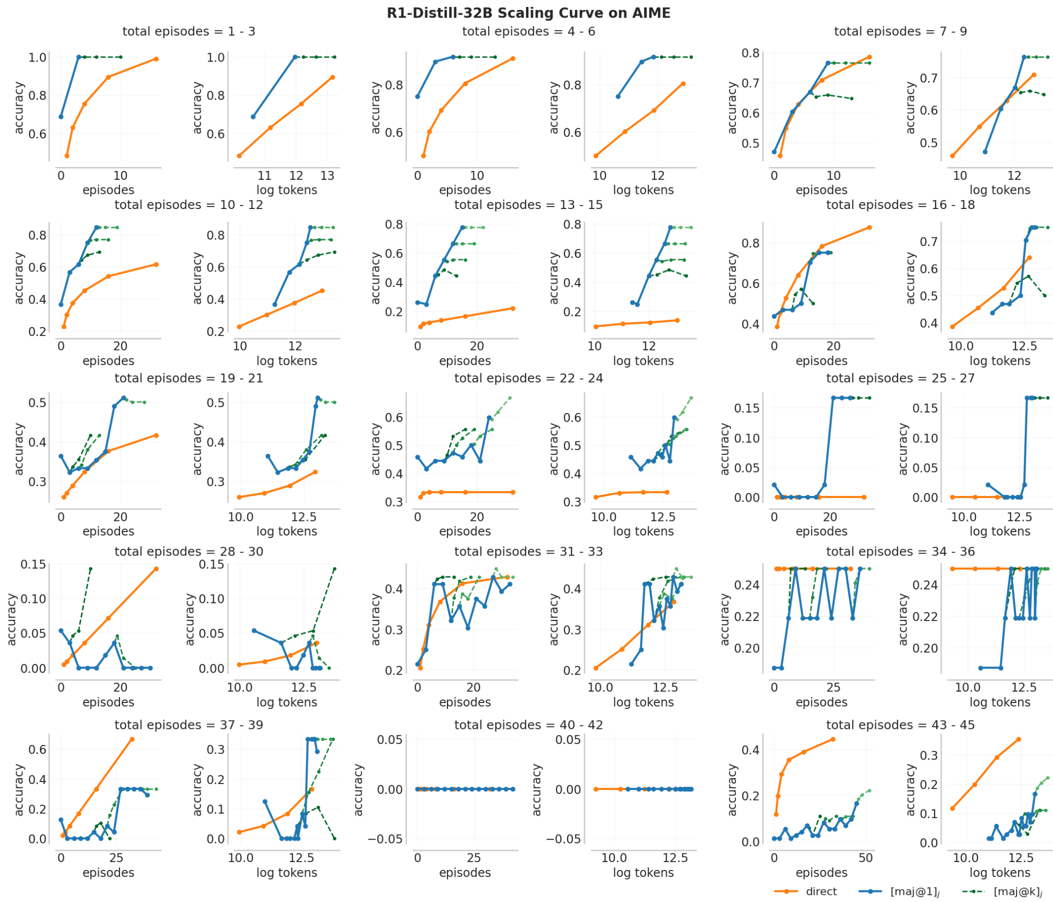

Figure 14: **R1 scaling curve on AIME 2024 across different episodes**. We compare scaling up R1 compute with **direct** pass@k for k = 1, 2, 8, 16, 32 against $[\textbf{maj@p}]_j$ for p = 1, 2, 8 and varying levels of j.

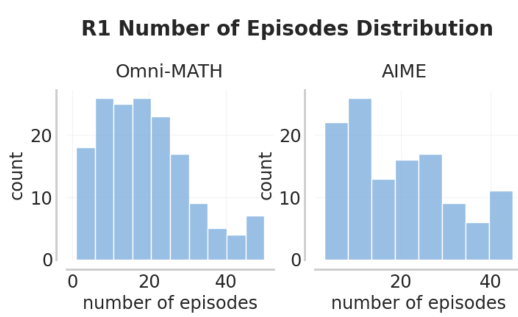

Figure 15: **Distribution of the number of episodes generated by R1 responses on AIME and Omni-MATH**.