B-STAR: MONITORING AND BALANCING EXPLORATION AND EXPLOITATION IN SELF-TAUGHT REASONERS

Anonymous authors
Paper under double-blind review

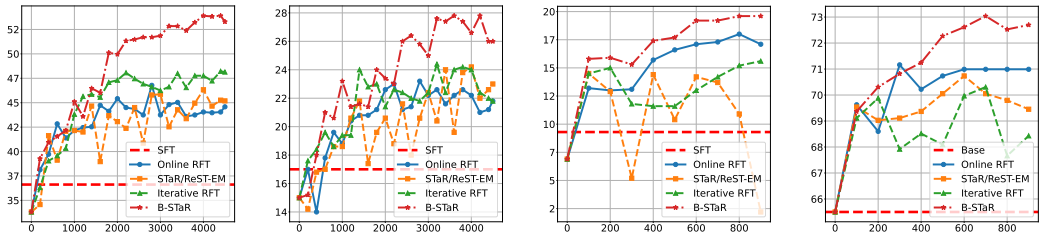
ABSTRACT

In the absence of extensive human-annotated data for complex reasoning tasks, self-improvement – where models are trained on their own outputs – has emerged as a primary method for enhancing performance. Recently, the approach to self-improvement has shifted toward a more dynamic, online fashion through iterative training processes. However, the critical factors underlying the mechanism of these self-improving methods remain poorly understood, such as under what conditions self-improvement is effective, and what are the bottlenecks in the current iterations. In this work, we identify and propose methods to monitor two pivotal factors in this iterative process: (1) the model's ability to explore and generate high-quality responses among multiple candidates (*exploration*); and (2) the reliability of external rewards in selecting the best responses from the generated outputs (*exploitation*). These factors are inherently moving targets throughout the self-improvement cycles, yet their dynamics are rarely discussed in prior research – It remains unclear what impedes continual model enhancement after only a few iterations. Using mathematical reasoning as a case study, we begin with a quantitative analysis to track the dynamics of exploration and exploitation, discovering that a model's exploratory capabilities rapidly deteriorate over iterations, and the effectiveness of exploiting external rewards diminishes as well due to shifts in distribution from the original policy. Motivated by these findings, we introduce B-STAR, a **S**elf-**Ta**ught **R**easoning framework that autonomously adjusts configurations across iterations to **B**alance exploration and exploitation, thereby optimizing the self-teaching effectiveness based on the current policy model and available rewards. Our experiments in mathematical reasoning demonstrate that B-STAR not only enhances the model's exploratory capabilities throughout training but also achieves a more effective balance between exploration and exploitation, leading to superior performance. Crucially, this work deconstructs the opaque nature of self-training algorithms, elucidating the interpretable dynamics throughout the process and highlighting current limitations for future research to address.

1 INTRODUCTION

Large language models possess advanced reasoning capabilities such as mathematical problem-solving (Cobbe et al., 2021) or coding challenges (Chen et al., 2021). However, the challenge of acquiring extensive, high-quality human-curated datasets remains a significant barrier to further enhancing these reasoning abilities. As tasks grow in complexity, the reliance on human-generated data becomes increasingly unsustainable, necessitating alternative approaches to training.

To tackle this issue, methods rooted in the concept of "Self-Improvement" (Huang et al., 2022), such as STaR (Zelikman et al., 2022), RFT (Yuan et al., 2023), and ReST (Gulcehre et al., 2023; Singh et al., 2023), provide more cost-effective and scalable solutions. Self-Improvement follows an iterative process where the model generates responses, from which the better are selected to create higher-quality data for further refinement (Hosseini et al., 2024). This continuous loop allows the model to improve its performance over time, reducing the need for large amounts of human-generated data. Ultimately, this approach enhances the model's ability to handle complex reasoning tasks, pushing the limits of its capabilities (Havrilla et al., 2024).



(a) GSM8K (b) MATH (c) APPS (d) ARC-C

Figure 1: [ARC-C results are added during rebuttal] Pass@1 accuracy over training steps on GSM8K, MATH, APPS, ARC-Challenge. We compare multiple baselines, including SFT, Online RFT, STaR/ReST-EM, and Iterative RFT, against our proposed B-STAR. For ARC-Challenge, we starts from the Mistral-7b-instruct model, denoted as "Base" in the figure.

Despite significant advancements, we still lack a deep understanding of the key factors that drive successful self-improvement and the internal optimization mechanisms remain largely opaque. Understanding the critical components and bottlenecks of these self-improving methods is particularly important given that performance of current self-improving approaches does not scale well with increased compute and saturates very quickly after merely 3 to 5 iterations (Singh et al., 2023; Wu et al., 2024). In this paper, we address the following questions: (1) What key factors play a decisive role in the self-improvement process? (2) How can these factors be used to analyze the limitations of current self-improvement methods from a unified perspective? and (3) How can we leverage these factors to guide the self-improvement process, ultimately maximizing performance gains?

To this end, we identify two crucial capabilities of the model during its self-improvement process: (1) the model's ability to generate correct and diverse solutions among multiple generated candidates (Singh et al., 2023; Wang et al., 2024a), and (2) the effectiveness of external rewards (e.g., a reward model or final answer supervision) in selecting high-quality solutions from these candidates (Wang et al., 2024b; Sun et al., 2024). Connecting with traditional RL terminology, we correspond the two capabilities to *exploration* and *exploitation* respectively. Intuitively, these two factors are dynamic, evolving throughout the training process, a phenomenon that remains underexplored despite its critical importance. In this work, we first conduct empirical analysis to quantitatively monitor the dynamics of exploration and exploitation during iterative training processes. We observe that both capabilities may stagnate or even decline, and imbalances between them can hinder the model's ongoing improvement.

Motivated by these insights, we propose a novel approach for self-improvement that automatically monitors and balances these dynamic factors to optimize the use of the current policy and reward. This involves adjusting configurations that influence exploration and exploitation, such as sampling temperature and reward thresholds. These configurations are adaptively modified throughout training in terms of our proposed metric, *query effect*. This new metric assesses the potential of a query based on the current model's exploration and exploitation capabilities, and our method automatically balances exploration and exploitation behaviors to maximize the average query effect scores. We refer to this method as B-STAR, a Balanced Self-Taught Reasoner.

The experimental results from mathematical problem-solving and coding challenges demonstrate that B-STAR significantly surpasses other self-improvement methods through balanced exploration and rewarding. For instance, B-STAR achieves a nearly 5 point increase absolutely in Pass@1 on both GSM8K and MATH, surpassing various self-improving variants while maintaining a steady upward trajectory, as depicted in Figure 1. Furthermore, we demonstrate that exploration-related metrics, such as Pass@32, are continuously improving without any notable degradation as in other baselines.

2 Monitoring Exploration and Exploitation in Self-Improvement

2.1 Background: Self-Improvement

Given a pre-trained model M_0 and a training set $D = (x_i, y_i)_{i=1}^N$, where x_i denotes the training queries and y_i their responses, the goal of self-improvement is to iteratively generate high-quality responses from the current model and update the model itself with such synthetic data. Let T represent

the total number of iterations, with the model at the start of the t-th iteration denoted as M_{t-1}. In the first iteration, M_0 is typically fine-tuned on the initial dataset D, then each subsequent iteration involves three critical steps generally (Yuan et al., 2023; Gulcehre et al., 2023):

(1) Generating: For each query x_i, the model M_{t-1} generates K candidate responses, forming a new, self-generated dataset.

(2) Rewarding (Verifying): A reward function $r(x, y)$ is applied to score and select the high-quality responses from the self-generated dataset. This reward can be binary and utilize additional supervision, for example, in problem-solving tasks in the math and code domains, it is common to match the final answer or unit tests pass result as the binary feedback (Yuan et al., 2023; Chen et al., 2022). In a more sophisticated case, $r(x, y)$ can be parameterized by a reward model outputting continuous scores, using outcome-based reward models (ORMs) (Li et al., 2022) or process-based reward models (PRMs) (Uesato et al., 2022; Lightman et al., 2023; Havrilla et al., 2024).

(3) Improving: The selected dataset is then used to update M_{t-1}, producing the updated model M_t. To differentiate the generation model M from the reward model, M is also referred to as the policy model following RL literature. In problem-solving tasks that we are going to focus on in this paper, SFT loss is commonly used in the Improving step due to its robustness and scalability (Pang et al., 2024; Dubey et al., 2024), as more sophisticated RL losses can be unstable to optimize and scale up. When SFT loss is adopted, the Rewarding step aims to reject some responses and use the remaining for training, thus this process is also referred to as rejection fine-tuning (RFT, Yuan et al. (2023)).

Discussion on Online Learning. The iterative procedure described above can be contextualized within the reinforcement learning framework (Singh et al., 2023), and the iterative design shifts the vanilla offline training towards a more dynamic online variant – when iteration intervals are short and the optimizer is inherited between iterations, the training essentially transforms into a fully online learning algorithm. For example, PPO (Schulman et al., 2017), a prevalent online RL algorithm, exemplifies iterative training with small iteration intervals. Iterative RFT implementations in previous works typically adopt long iteration intervals where each iteration processes all the queries (Zelikman et al., 2022; Sun et al., 2024). Sometimes, these implementations opted to restart the training from the initial checkpoint rather than from the last saved model (Zelikman et al., 2022; Singh et al., 2023). However, we argue that always starting from the beginning is not scalable for large datasets particularly in a streaming setting, instead, a more continuous training approach aligning more closely with RL principles is preferable. Transitioning from offline to online training, Online RFT (Shao et al., 2024) has demonstrated its superiority compared to traditional offline RFT methods. Compared to conventional iterative RFT, online RFT switches iterations more frequently so that the synthetic data is always on-policy. In this study, we explore online RFT as our primary framework for self-improvement, choosing a moderate iteration interval for more stable training. In § 4.2, we will demonstrate that our online RFT approach surpasses conventional iterative RFT training.

2.2 The Critical Factors – Exploration and Exploitation

To maximize the gains from self-improvement training and even lift model's ability fundamentally from its own outputs, the key is to achieve scalable self-improvement training, where the model performance is able to scale up with increased compute invested into the training algorithm. However, all previous works show quick saturation after merely 3-5 self-improvement iterations (Singh et al., 2023; Wu et al., 2024), where it is hypothesized that the model's own outputs can only lead to limited gains. In this work, we seek to dive deeper into the currently opaque process of self-improvement, to understand the critical factors that make self-improvement successful or failed.

Intuitively, for a certain iteration of training, we argue that two high-level conditions must be met for the model to make progresses: (1) **Diverse Exploration for High-Quality Responses:** When multiple candidates are sampled from the model, a portion of them must be high-quality responses. This is particularly important for queries where the model fails to produce satisfactory outputs using greedy decoding. Models often learn the most from queries they struggle with, and achieving this condition requires the model to explore sufficiently diverse outputs, enabling it to generate responses that cannot be reached through greedy decoding. (2) **Effective Reward Function Discrimination:** The reward function $r(x, y)$ must reliably distinguish high-quality candidates from lower-quality ones.

If either of these conditions is unmet—such as when the model produces responses overly similar (i.e. lack of diversity), or when the reward function fails to identify high-quality responses—the self-improvement will be limited on the gains.

Exploration and Exploitation are Moving Targets. Both exploration and exploitation are dynamically influenced by the policy model during the self-improvement process. On one hand, after multiple iterations, the policy model may overfit the task, failing to explore diverse responses and instead generating highly similar outputs (i.e., a lack of exploration). Training on these highly similar responses is unlikely to yield significant improvements. On the other hand, if the model generates excessively diverse responses, resulting in a distribution that deviates significantly from the reward model, it becomes challenging for the reward model to reliably distinguish high-quality responses (i.e., a lack of exploitation). Thus, maintaining a dynamic balance between exploration and exploitation is essential throughout the self-improving process. However, such dynamics are rarely discussed in prior research, while Wu et al. (2024) reveal a model's generation diversity tends to decline over the course of self-training, indicating a decrease in exploration capabilities. Next, we propose methods to quantify exploration and exploitation, enabling us to monitor their dynamics during training and deepen our understanding of the mechanisms underlying self-improvement.

Quantifying Exploration and Exploitation. In this work, we mainly focus on complex problem-solving tasks in the math and coding domains, where the correctness of the responses can be easily verified on labeled datasets – in the math domain, it is common to verify whether the generated final answer matches the ground-truth one,[1] while in code domain we typically verify whether the generated code passes the given unit tests. This property makes it easier to quantify exploration and exploitation, for which we detail the metrics below:

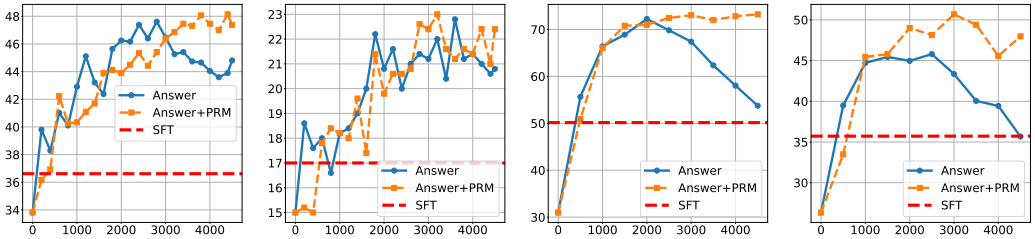
- *Exploration:* Pass@K, which measures whether there is at least one correct response during K sampled candidates, is a straightforward metric to measure exploration, as it directly reflects whether the model is able to explore correct solutions. However, Pass@K may be noisy as it only counts one correct response, while it is desirable to assess whether the model can explore more than one correct response as well. To this end, we propose to track Pass@K-S as well, which measures whether there are at least S correct responses among K sampled candidates. Pass@K-S serves as a more stable proxy to exploration than Pass@K. Pass@K is essentially Pass@K-1 following such a definition. Besides Pass@K and Pass@K-S, we also track diversity of the generations using Distinct Equations proposed by Wu et al. (2024), which measures the proportion of unique equations among all correct generated responses.
- *Exploitation:* Best-of-K accuracy measures whether the top one response ranked by the reward function is correct or not, which directly reflects how well the reward can potentially select one good response. Since it is typically required to select multiple responses rather than one in self-improving training, we are interested in the reward's exploitation to select multiple responses as well. To this end, we come up with the Reward@K-S metric, which measures whether the top S candidates ranked by the reward are all correct or not. Best-of-K accuracy is a special case of Reward@K-S when S is equal to 1. One may think of Reward@K-S would be equal to Pass@K-S if only the final answer is used to select responses, then Reward@K-S may not be useful in this case. However, we emphasize that the exploitation metrics are mainly used to measure the effectiveness when additional reward models are integrated, as we will show next in §2.3 that our reward function combines final answer supervision and a reward model.

Next, we conduct a case study to dive into self-improving training through tracking these metrics.

2.3 Dynamics of Exploration and Exploitation – A Case Study in Mathematical Problem Solving

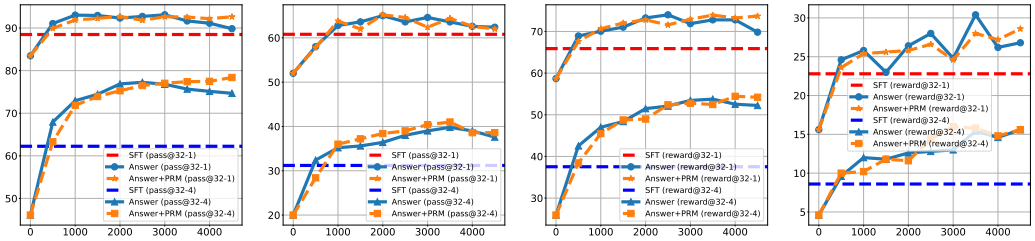
In this section, we perform a case study to analyze the dynamics of exploration and exploitation in a mathematical reasoning task. Specifically, we follow Singh et al. (2023) to adopt MATH (Hendrycks

[1] Strictly speaking, the response may contain incorrect steps even though the final answer is correct, we do not further distinguish this difference following others (Zelikman et al., 2022; Singh et al., 2023) as it is not the focus of this work.



(a) Pass@1 on GSM8K (b) Pass@1 on MATH (c) Diversity on GSM8K (d) Diversity on MATH

Figure 2: Pass@1 and Diversity over training steps on GSM8K and MATH



(a) Pass@K-S on GSM8K (b) Pass@K-S on MATH (c) Reward@K-S GSM8K (d) Reward@K-S MATH

Figure 3: Pass@K-S and Reward@K-S over training steps on GSM 8K and MATH

et al., 2021) training set as the training data, evaluating on the test split of GSM8K (Cobbe et al., 2021) and MATH.

Setup. As introduced in §2.1, we adopt the online RFT training framework in our implementation due to its superior efficiency over conventional iterative RFT. We use Mistral-7B (Jiang et al., 2023) as our base model. We run the SFT baseline on MATH for 3 epochs, and use the checkpoint at the first epoch as the initial checkpoint to run self-improving training. We experiment with two different types of reward functions in the dynamics analysis:

- *Answer:* we just match the predicted answer with the ground-truth final answer and keep the responses with correct answers, following (Singh et al., 2023).
- *PRM:* we train a process reward model (PRM) using the approach in Wang et al. (2024b) based on Mistral-7B. Then we combine the final answer reward and the PRM reward as:

$$r = \mathbb{1}(\hat{a} = a*) + r_{prm}(x, \hat{y}), \tag{1}$$

  where $\hat{a}, a*$ are the predicted answer and the ground-truth answer respectively, $\mathbb{1}(\cdot)$ is the indicator function, $\hat{y}$ is the predicted solution. $r_{prm}(\cdot)$ is the PRM score. Since PRM is designed to score every step of the solution, we choose the minimal score across all steps as the score for the entire solution, following Lightman et al. (2023). We only keep responses with $r > \tau$ to train the model, and τ is a threshold. We found $\tau = 0$ is a good hyperparameter in our early trials with different thresholds, thus we keep $\tau = 0$ in all the experiments on dynamics analysis. In our preliminary experiments, we also tried another reward alternative where we prefilter responses with the final answers and then select the remaining ones by PRM, but we found that underperforms the reward function in Eq. 1.

We train all methods for 4500 training steps. For our online RFT, we adopt an iteration interval with 500 steps, which means the online RFT training has a total of 9 iterations, significantly larger than the typical iteration numbers 3-5 in previous works (Singh et al., 2023). We sample 32 candidate solutions per query during training with a temperature of 1.0. We include the SFT baselines that is trained for 3 epochs on MATH. Please see Appendix A for more setup details.

Observation 1: Self-improving training increases accuracy significantly, while PRM slightly helps. Compared to SFT, the Self-Improvement method significantly enhances the model's performance in generating accurate solutions via greedy decoding. As illustrated in Figure 2 Left, both methods achieved an approximate 10 acc increase in pass@1 on GSM 8K, and about a 5 acc increase

on MATH, compared to direct SFT. However, we observed that the performance gains from both methods gradually stagnated as iterations progressed, with pass@1 plateauing or even declining after around 3,000 steps when only the final answer provides the reward. After integrating PRMs, about 3 point again is achieved compared to the final answer reward on GSM8K, and the decline trend of accuracy is mitigated. In contrast, the combined reward did not show a significant advantage on MATH. We hypothesis that this is because the MATH problems are too difficult for the 7B reward model to discriminate solutions well.

Observation 2: Exploration decreases over training and PRM helps retain: As shown in Figure 2 right, the diversity metric on both datasets decreases dramatically, a similar phenomenon observed in Wu et al. (2024). Surprisingly, the combined answer and PRM reward is able to overcome the declining trend and retain exploration. We suspect that this is because filtering solutions solely based on answer correctness often leads to many homogeneous results. In contrast, the fine-grained reward strategy promotes the selection of high-quality steps, which indirectly helps preserve solution diversity. Investigating the Pass@K-S metrics in Figure 3 Left, Pass@K-S increases in the beginning, which implies the model's exploration is improved first, yet then Pass@K-S starts to decrease if only final-answer reward is adopted. Critically, the Pass@K-1 accuracy even falls close to the SFT baseline. The integration of PRM is able to improve Pass@K-S on GSM8K slightly, but ineffective on MATH potentially due to the limited capability of the reward model. The plateau pattern of exploration is not a good sign, as it implies that the model may not learn new things to explore better. Given that our reward is fixed during training, the model's ability will stop increasing quickly if exploration stops improving.

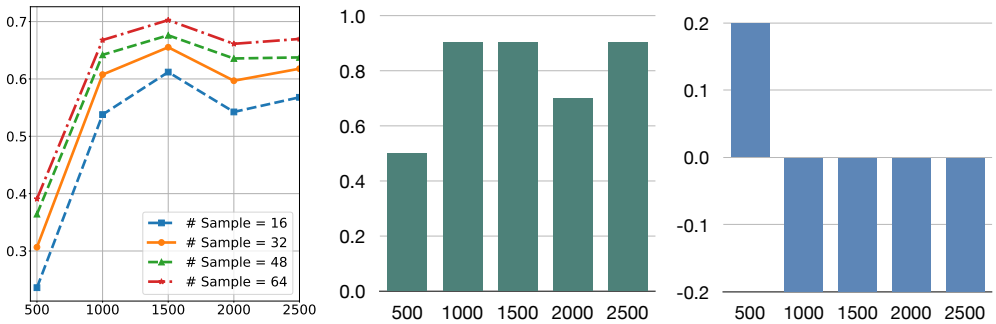
Observation 3: Exploitation saturates on GSM8K while keeps improve on MATH: As shown in Figure 3 Right, we observe different behaviors of exploitation on GSM8K and MATH – Reward@K-S shows saturation on GSM8K, yet continually improves on MATH, potentially because we are training with the MATH dataset. We note that our reward is static during training as we do not update it or change how we select responses, then exploitation performance is closely related to exploration from the policy model, which is a moving target and unfortunately, from our previous analysis, exploration of the model does not show continual improvement in all cases. We hypothesize this is the key factor that bottlenecks self-improving training. Exploration is related to configurations such as how we sample responses from the model, how many samples to draw. Similarly, exploitation depends on how we utilize the reward to select responses. While all previous works treat these configurations as static during training as far as we know, can we adjust them dynamically so that exploration and exploitation better fit each other? We study this question next.

3 B-STaR – Balanced Self-Taught Reasoners

In this section, we first introduce a metric designed to guide us to balance exploration and exploitation, then we analyze its relationship with various configurations. Finally, we present our full algorithm that automatically adjusts exploration and exploitation abilities.

3.1 Query Effect

§2 highlights the importance of a model's exploration and exploitation capabilities for self-improvement, emphasizing their continuous evolution during training. Now we seek for a metric that could provide us information on the interplay of these two factors. Intuitively, we hope the model can explore more high-quality responses and the reward can select them out. As a result of this interplay, given a query, we expect two conditions to be satisfied so that the selected responses can contribute effectively to training: (1) the high-quality responses among the selected ones cannot be too small, otherwise we do not have enough good data for training; (2) the percentage of the high-quality responses among the selected ones must be large enough, otherwise we may mix too many bad data points into the training data. The former focuses on the quantity of high-quality responses selected, whereas the latter prioritizes the ratio (but not the absolute quantity) of the high-quality responses among all selected responses. For example, if we select only 2 response and they are correct, then the ratio is 100% while the absolute number is only 2, which cannot give us enough training data; if we select all 64 candidates, suppose 16 of them are correct and others are wrong, then we have 16 high-quality responses, but the ratio is only 25% – feeding many incorrect responses into training is



(a) Query Effect Score (b) Best Temperature (c) Best Threshold

Figure 4: **Left:** the average query effect scores on 600 MATH training queries at different training steps, varying the number of samples to draw per query; **Middle:** the best sampling temperature to achieve the max average query effect scores at different training steps; **Right:** the best reward threshold to achieve the max average query effect scores at different training steps;

undesirable. Therefore, these two capture different aspects of the data. Following these two intuitions, we propose a metric, *query effect*, to measure the potential positive effect of the responses selected from a query, given the exploration- and exploitation-related configurations. We detail it below.

For each query x_i, let n_i represent the total number of selected responses for it, and n'_i denote the number of unique, correct responses among them, then the query effect is defined as:

$$qe_i = \min\left(\frac{n'_i}{n^\star},\ 1\right) \cdot \frac{n'_i}{n_i}, \tag{2}$$

where $(\frac{n'_i}{n^\star},\ 1)$ is a discount factor that encourages the number of correct solutions to be larger than a pre-specified parameter $n^\star$ – there is no discount when we have more than $n^\star$ correct responses. We impose 1 as the cap, rather than encouraging n'_i to be as large as possible, because otherwise the number of responses among queries will be severely imbalanced (Tong et al., 2024), where the easy queries will occupy most of the correct responses to maximize the average query effect. The second term, n'_i/n_i, is the ratio of the correct responses. We note that this ratio is always 100% if only the final-answer reward is used. $n^\star$ is the only hyperparameter in this metric, and it roughly implies the correct responses that we aim to have per query. Suppose we aim to select N samples per iteration, and each iteration we feed in M queries where $N > M$, then we simply decide $n^\star = \lceil \frac{N}{M} \rceil$. As N and M are just general training hyperparameters related to data loader, we never tune them in the paper and just set them as reasonable numbers as we will describe later. This means, *the query effect metric does not introduce additional hyperparameters for us to tune*.

Our goal is to adjust exploration and exploitation to maximize the average query effect of a subset of training queries – the max value of qe_i is 1 and the average query effect is maximized when all the selected responses are correct and there are at least $n^\star$ correct responses for each query.

3.2 Configurations of Exploration and Exploitation

We explore various configurations that directly relate to exploration and exploitation. Below we introduce them and analyze their influence on the average query effect scores. Specifically, we obtain the policy model checkpoints and reward model checkpoints of different iterations from the online RFT with final answer + PRM reward run, then we apply different configurations to these checkpoints, and compute the average query effect scores on 600 randomly sampled MATH training queries.

Exploration – sample size. There are two configurations which affect exploration: the number of samples to draw per query and the sampling temperature. We begin by examining the impact of sample size, where the sampling temperature is fixed at 1.0, the reward threshold is set to 0.0, and we adjust the sample sizes to 16, 32, 48, and 64, using models from the 1st, 3rd, 5th, 7th, and 9th iteration. The results, shown in Figure 4(a), indicate that *increasing the sample size always enhances query effect*. This suggests that we should always use the max samples size allowed by sampling budget.

Algorithm 1 B-STAR Algorithm

Input: Total iterations I, initial policy model P_0, reward model RM, full dataset $\mathcal{D}$, temperature set $\mathcal{T}$, sample size k, reward threshold set Θ. Initialized optimizer $\mathcal{O}_0$, learning rate scheduler $\mathcal{L}_0$
Output: Final updated model P_I

```
for i = 1 to I do
    // Determine exploration and exploitation configurations
    Choose sampling temperature t_i ∈ T to maximize the average query effect score (§3.1)
    Choose sampling temperature τ_i ∈ Θ to maximize the average query effect score (§3.1)
    // Generate
    Generate dataset D_i by sampling M queries from D, and then sampling k responses per query with the
    adjusted temperature t_i
    // Improve
    Annotate reward r for D_i, then only keep the samples with reward larger than τ_i
    Update the policy model with the selected data, P_i ← P_{i-1}
    // We inherit optimizer and learning rate scheduler for online RFT
    O_i ← O_{i-1}, L_i ← L_{i-1}
end for
```

Exploration – temperature. We next examine how sampling temperature affects query effect. With the reward threshold fixed at 0.0 and a sample size of 32, we adjust the sampling temperature to 0.5, 0.7, 0.9, and 1.1. In Figure 4(b) we show the best temperature we obtained that maximizes the average query effect at different iterations (training steps). It reveals the optimal temperature is different as training progresses – lower temperature is preferred in the beginning while higher temperature is better later on. This phenomenon can be explained by the model's shifting limitations during training. In the early stages, the model's ability to generate correct solutions is weak, so lower temperatures help to sample more accurately (Yang et al., 2023). As training advances, the challenge shifts to preserving diversity in the generated solutions, requiring higher temperatures to ensure broader sampling.

Exploitation – threshold. We investigate the impact of reward thresholds on query effect, the configuration that decides how the reward exploit the samples. In our experiments, we fix the sampling temperature at 1.0 and the sample size at 32, while varying the reward threshold and selecting only solutions that exceed the threshold. Figure 4(c) presents the optimal threshold that maximizes the average query effect score. It indicate that higher threshold is preferred in the beginning and it may need to decrease subsequently. Intuitively, that suggests that we should adopt more rigorous filtering in the beginning when the model is weaker, and relax the threshold a bit when the model grows stronger.

3.3 B-STAR

Building on the findings from §3, we propose B-STAR, shorten for Balanced Self-Taught Reasoners, a method that maximizes the average query effect by dynamically adjusting configurations to balance exploration and exploitation. Specifically, § 3.2 shows that the sample size is the best to be chosen as large as possible obeying our sampling budget, while temperature and threshold may need to be dynamically adjusted. Therefore, we adjust temperature and threshold automatically at every iteration, to maximize the average query effect. Notably, we only need to compute the query effect score on a small subset of training queries to decide the balanced configurations, thus incurring negligible additional costs compared to the baselines. For example, we only use 600 MATH queries in our experiments. The full algorithm is summarized in Algorithm 1.

4 MAIN EXPERIMENTS

4.1 SETUP

We evaluate B-STAR's effectiveness in enhancing self-improvement for mathematical problem-solving, coding challenges and commonsense reasoning. For mathematical problem-solving, we largely maintain the experimental setup from § 2.3, including datasets and baselines. Based on findings from § 3.2 that show a monotonic increase in query effect with sample size, we set sample size to 64 for all methods. We vary temperature from 0.5 to 1.2 in 0.1 increments and reward

Methods	GSM 8K			MATH			APPS			ARC-C
	P@1	P@32	P@32-4	P@1	P@32	P@32-4	P@1	P@32	P@32-4	P@1
SFT	36.6	88.5	62.2	17.0	60.8	31.2	9.3	43.5	25.5	—
Rest-EM (w/o RM)	40.5	89.9	69.8	22.8	60.0	33.6	14.5	43.9	28.2	70.7
Rest-EM (w/ RM)	46.3	90.7	72.2	24.2	62.8	37.4	—	—	—	—
Iterative RFT (w/o RM)	42.8	88.9	71.3	24.2	63.4	38.2	15.2	44.3	28.0	70.3
Iterative RFT (w/ RM)	46.6	90.2	74.9	24.4	62.6	39.0	—	—	—	—
Online RFT (w/o RM)	44.0	88.1	69.7	23.0	57.2	38.2	17.3	45.8	27.8	71.2
Online RFT (w/ RM)	46.8	91.4	76.5	23.2	62.6	39.2	—	—	—	—
B-STAR	**53.8**	**93.6**	**81.0**	**27.8**	**67.2**	**42.2**	**19.6**	**49.3**	**30.7**	**73.0**

Table 1: [ARC-C results are added during rebuttal] Comparison of self-improvement methods across MATH, GSM 8K, APPS and ARC-Challenge, showing highest Pass@1 (P@1) scores with corresponding Pass@32 (P@32) and Pass@32-4 (P@32-4). Methods include variants with and without a reward model ("w/ RM" and "w/o RM"). Online RFT (w/o RM) is the Answer baseline, while Online RFT (w/ RM) is the Answer+PRM baseline.

Step	**500**	**1000**	**1500**	**2000**	**2500**	**3000**	**3500**	**4000**	**4500**
Sample Number	64	64	64	64	64	64	64	64	64
Temperature	0.5	0.8	0.9	1	1.1	1.1	0.9	1.1	1.1
Reward thresholds	0	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1
Query Effect	0.470	0.538	0.589	0.621	0.646	0.660	0.673	0.678	0.679

Table 2: Dynamic changes in B-STAR hyperparameters.

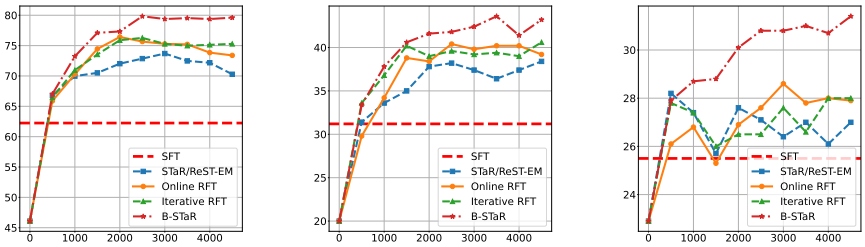
threshold from -1.0 to 1.0 in 0.1 increment. Throughout the self-improvement process, we use Pass@1, Pass@K-S, and Reward@K-S metrics to track changes in the performance and model's exploration and exploitation capabilities.

To evaluate the generalization of B-STAR, we further conduct experiments on coding challenges and commonsense reasoning. Specifically, for coding challenge, following Singh et al. (2023), we adopt the APPS (Hendrycks et al., 2021) dataset for both training and testing. To balance the number of solutions per question, we sample 5 solutions from the original APPS training set forming a dataset with 13K examples. We use Llama-3-8B (Dubey et al., 2024) as our base model, keeping the rest of the settings consistent with those of the math domain. For baselines, we uniformly sample 32 candidate solutions per query with a temperature of 0.4. For B-STAR, we explore temperatures 0.4 to 1.1 in 0.1 increment to determine the optimal configuration. We do not apply reward models to the coding task and instead use unit tests as the binary reward, which means only the sampling temperature is automatically adjusted in B-STAR.

For commonsense reasoning, following Pang et al. (2024), we conduct experiments on ARC-Challenge (Clark et al., 2018), a dataset consisting of multiple-choice science questions designed to evaluate commonsense reasoning beyond mathematics and coding challenges. We start with the Mistral-7b-instruct model and omit the SFT stage due to the absence of the Chain-of-Thought (CoT) data for this dataset. Other configurations, such as sample size and temperature, are the same as those used in the coding tasks. The ground-truth answer serves as the binary reward, and we report only Pass@1 results for the ARC-Challenge dataset. Given the constrained answer space inherent to multiple-choice questions, Pass@K and Pass@K-S metrics (where $K > 1$) yield no additional insights and are therefore excluded.

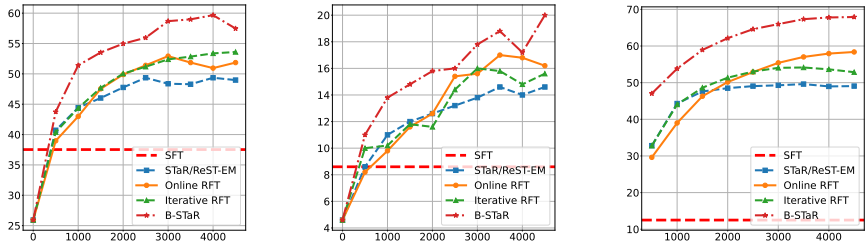
4.2 RESULTS

Mathematical Reasoning. Table 1 provides a comprehensive comparison of B-STAR with various self-improvement methods, including Rest-EM, Iterative RFT, Online RFT, and their reward model variants. The results show that B-STAR consistently achieves higher pass@1 scores across both the GSM 8K and MATH datasets, highlighting its ability to effectively steer the model toward correct solutions via greedy decoding. Moreover, B-STAR demonstrates significantly better pass@K-S values, reflecting an enhanced exploration capacity that facilitates the generation of a wider range of high-quality responses. Notably, Online RFT outperforms predominantly offline methods like Rest-EM, illustrating that dynamic, on-policy approaches strike a more effective balance between learning efficiency and performance gains.



(a) GSM8K (b) MATH (c) APPS

Figure 5: Pass@K-S over training steps on GSM8K, MATH and APPS.



(a) GSM8K (b) MATH (c) Query Effect

Figure 6: Reward@K-S over training steps on GSM8K (left), MATH (middle). Query Effect over training steps (right).

In Figures 1, 5, and 6, we illustrate the dynamic evolution of B-STAR and Online RFT throughout the self-improvement process. Figure 1 reveals that B-STAR significantly outperforms other approaches in generating accurate responses through greedy decoding. In addition, Figures 5 and 6 demonstrate that B-STAR effectively balances exploration and exploitation during the self-improvement process, enabling the model to generate a broader set of accurate solutions while efficiently integrating feedback from the reward model. This balance is reflected in the higher and more stable Pass@K-S and Reward@K-S metrics for B-STAR across both datasets.

Table 2 showcases the configurations automatically tuned by B-STAR at different training iterations, along with their corresponding query effects. During the initial training stages, B-STAR reduces the sampling temperature to lower values, such as 0.5. As training progresses, the temperature gradually increases, aligning with our observations in Section 3.2. Moreover, B-STAR dynamically adjusts the reward thresholds to moderate range, maximizing the query effect and effectively leveraging feedback from the reward model.

Coding and Commonsense Reasoning. As shown in Figure 1 and 5, all the self-improvement methods exhibit significant performance improvement after the first iteration. However, as the number of iterations increases, the growth trends of other baseline methods slows down and eventually stagnate. In contrast, B-STAR maintains a substantial growth rate and consistently outperforms the two baselines, SFT and RFT. This suggests that balancing exploration and exploitation is crucial for achieving stable and efficient self-improvement, further validating the generalizability of B-STAR.

5 Discussion

In this paper, we conduct a comprehensive investigation into the intrinsic mechanisms of self-improvement, identifying two critical factors: (1) the model's ability to explore and generate high-quality responses from a diverse set of candidates (exploration), and (2) the reliability of external rewards for selecting optimal responses (exploitation). Through quantitative experiments, we track the evolution of these capabilities throughout the self-improvement process. Our findings indicate that balance of them is required. To this end, we propose B-STAR, a method that dynamically adjusts configurations during the self-improvement process to maintain a balance between exploration and exploitation. Our experiments in mathematical reasoning and coding challenges demonstrate that B-STaR significantly improves the performance.

REFERENCES

Sungyong Baik, Myungsub Choi, Janghoon Choi, Heewon Kim, and Kyoung Mu Lee. Meta-learning with adaptive hyperparameters. *Advances in neural information processing systems*, 33: 20755–20765, 2020.

Atilim Gunes Baydin, Robert Cornish, David Martinez Rubio, Mark Schmidt, and Frank Wood. Online learning rate adaptation with hypergradient descent. *arXiv preprint arXiv:1703.04782*, 2017.

Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. Codet: Code generation with generated tests. *arXiv preprint arXiv:2207.10397*, 2022.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

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

Alex Havrilla, Yuqing Du, Sharath Chandra Raparthy, Christoforos Nalmpantis, Jane Dwivedi-Yu, Maksym Zhuravinskyi, Eric Hambro, Sainbayar Sukhbaatar, and Roberta Raileanu. Teaching large language models to reason with reinforcement learning. *arXiv preprint arXiv:2403.04642*, 2024.

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. *arXiv preprint arXiv:2103.03874*, 2021.

Arian Hosseini, Xingdi Yuan, Nikolay Malkin, Aaron Courville, Alessandro Sordoni, and Rishabh Agarwal. V-star: Training verifiers for self-taught reasoners. *arXiv preprint arXiv:2402.06457*, 2024.

Jiaxin Huang, Shixiang Shane Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. Large language models can self-improve. *arXiv preprint arXiv:2210.11610*, 2022.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Hadi S Jomaa, Josif Grabocka, and Lars Schmidt-Thieme. Hyp-rl: Hyperparameter optimization by reinforcement learning. *arXiv preprint arXiv:1906.11527*, 2019.

Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. Making large language models better reasoners with step-aware verifier. *arXiv preprint arXiv:2206.02336*, 2022.

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

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Richard Yuanzhe Pang, Weizhe Yuan, Kyunghyun Cho, He He, Sainbayar Sukhbaatar, and Jason Weston. Iterative reasoning preference optimization. *arXiv preprint arXiv:2404.19733*, 2024.

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

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

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

Leslie N Smith. A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820*, 2018.

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 preprint arXiv:2403.09472*, 2024.

Yuxuan Tong, Xiwen Zhang, Rui Wang, Ruidong Wu, and Junxian He. Dart-math: Difficulty-aware rejection tuning for mathematical problem-solving. *arXiv preprint arXiv:2407.13690*, 2024.

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. *arXiv preprint arXiv:2211.14275*, 2022.

Evan Wang, Federico Cassano, Catherine Wu, Yunfeng Bai, Will Song, Vaskar Nath, Ziwen Han, Sean Hendryx, Summer Yue, and Hugh Zhang. Planning in natural language improves llm search for code generation. *arXiv preprint arXiv:2409.03733*, 2024a.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439, 2024b.

Ting Wu, Xuefeng Li, and Pengfei Liu. Progress or regress? self-improvement reversal in post-training. *arXiv preprint arXiv:2407.05013*, 2024.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. *arXiv preprint arXiv:2309.03409*, 2023.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*, 2023.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

Step	500	1000	1500	2000	2500	3000	3500	4000	4500
Sample Number	64	64	64	64	64	64	64	64	64
Temperature	0.65	0.75	1.05	0.95	1.05	0.85	1.05	1.15	1.05
Reward Thresholds	-0.02	-0.04	-0.09	-0.09	-0.14	-0.14	-0.14	-0.15	-0.06
Query Effect	0.500	0.557	0.591	0.626	0.652	0.665	0.679	0.682	0.684

Table 3: Finer-grained dynamic changes in B-STAR hyperparameters.

A EXPERIMENT SETUP FOR THE CASE STUDY

Datasets We use the MATH dataset for training and validate the model's mathematical reasoning ability using test sets from both the MATH (Hendrycks et al., 2021) and GSM 8K (Cobbe et al., 2021)datasets. For the MATH dataset, we follow previous setting (Lightman et al., 2023; Wang et al., 2024b; Sun et al., 2024) by using a subset of 500 representative problems (MATH500) as our test data. We uniformly sample an additional 500 problems for validation and use the remaining 4,000 problems from the MATH test set along with the original 7,500 training problems as our training data.

Implementation details For SFT, we use Mistral-7B (Jiang et al., 2023) as the base model with a learning rate of 5e-6, a batch size of 128, and train for 3 epochs. After the first epoch, the model (denoted as M_1) is used as the starting point for self-improvement. We then proceed with 9 iterations, where each iteration consists of 500 training steps with a batch size of 128. At the beginning of each iteration, we sample 32 candidate solutions for each query, using a temperature of 1.0.

For the Process Reward Model (PRM), we automatically generate process annotations following the MATH-Shepherd approach (Wang et al., 2024b). Using the SFT model trained for 1 epoch, we sample 15 solutions for each query in the training set. The SFT model trained for 3 epochs serves as the completer, decoding 8 solutions per step to annotate the sampled data. This process results in approximately 270 K process reward annotations. We train the reward model using the Mistral-7B base, with a learning rate of 2e-6, for 2 epochs. During rewarding, we use the lowest step score in the solution as the PRM Reward, normalize it to a range of [-1, 1] and combine it with a sparse reward to form the final reward score. We set the reward threshold to 0.0, selecting only those responses with final reward scores exceeding this threshold.

B DETAILS OF SELF-IMPROVEMENT

Supervised Fine-tuning Loss
Supervised fine-tuning (SFT) is employed to tailor a LLM to specific downstream tasks. Given a pre-trained model M_0 parameterized by θ, and a training set $D = \{(x_i, y_i)\}_{i=1}^{N}$, where x_i denotes the training queries and y_i their corresponding responses, the objective is to fine-tune the model. The training objective of SFT is to minimize the negative log-likelihood of the answers:

$$\mathcal{L}_{\text{SFT}}(\theta) = -\mathbb{E}_{(x,y)\sim\mathcal{D}} \log p(y \mid x; \theta) \tag{3}$$

Here, $p(y|x;\theta)$ represents the probability of generating answer y given the input query x, as determined by the model with parameters θ.

Greedy Decoding
Greedy decoding is a deterministic decoding strategy that selects the most probable token at each time step:

$$y_t = \arg\max_{y} p(y \mid x; y_{<t}; \theta) \tag{4}$$

Here, $p(y \mid x; y_{<t}; \theta)$ represents the probability of generating token y given the input query x and the tokens generated so far $y_{<t}$.

Sample Decoding
Sample decoding introduces randomness by sampling from the probability distribution over possible tokens at each step. Formally, at each time step, the next token y_t is sampled as:

$$y_t \sim p(y \mid x; y_{<t}; \theta) \tag{5}$$

This means that y_t is selected based on the likelihood given by the model, introducing variability and allowing for more diverse outputs.

Fixed Reward Function
A fixed reward function $r(x, y)$ is a predefined, static function that does not adapt based on the training process or model parameters. For instance, binary feedback (e.g., whether a math problem is solved correctly or a unit test passes) is an example of a fixed reward function.
The fixed reward function can be represented as:

$$r(x, y) = \begin{cases} 1, & \text{if } y \text{ satisfies a predefined condition (e.g., test passes),} \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

Reward Model
A trained reward function $r(x, y; \phi)$ is parameterized by ϕ and adapts based on the training process. These reward functions (such as output-based reward models (ORMs) or process-based reward models (PRMs)) are learned from data, where the model learns to assign continuous scores based on supervision signals.
The trained reward model can be represented as:

$$r(x, y; \phi) = f(x, y; \phi) \tag{7}$$

where $f(x, y; \phi)$ is a learned function (e.g., a neural network) that maps the input x and the response y to a continuous reward score.

Rejection Sampling Fine-tuning
Rejection Sampling Fine-tuning (RFT) first samples multiple outputs from the supervised fine-tuned LLMs for each query and then trains LLMs on the sampled responses with the correct answer. Formally, the objective of RFT is to maximize the following objectives:

$$\mathcal{L}_{\text{RFT}}(\theta) = \mathbb{E}_{x \sim \mathcal{D}} \left[\mathbb{I}(y) \log p(y \mid x; \theta) \right] \tag{8}$$

The indicator function $\mathbb{I}(y)$ is defined as:

$$\mathbb{I}(y) = \begin{cases} 1, & \text{if the answer of } y \text{ is correct,} \\ 0, & \text{if the answer of } y \text{ is incorrect.} \end{cases} \tag{9}$$

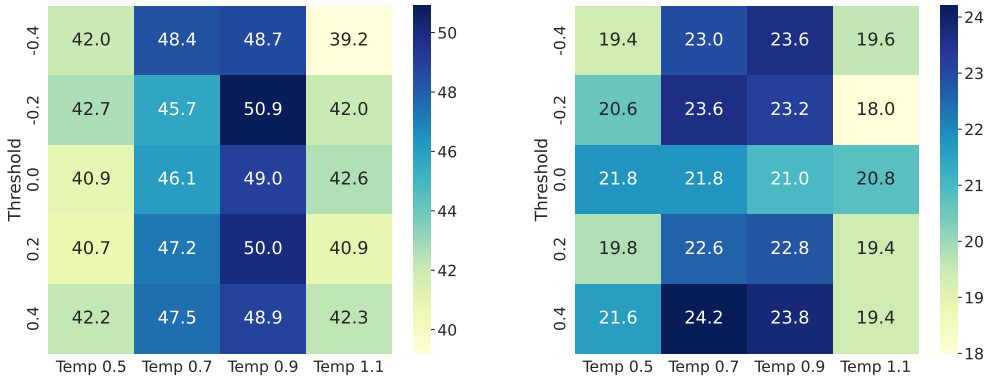
Our evaluation compares several baseline methods: STaR/ResT-EM, Iterative RFT, and Online RFT. The STaR/ResT-EM approach involves multiple iterations, where each iteration samples from the latest policy model but resets and retrains the model from scratch. In contrast, Iterative RFT builds on the previous iteration by inheriting the checkpoint and continuing training from that point. Online RFT extends this further by inheriting both the checkpoint and the full training state, enabling seamless progress across iterations. Additionally, we evaluate two variants: "without RM" (Answer) and "with RM" (Answer+PRM), as described in Section § 2.3.

C More Fine-Grained Configuration Adjustments

In Section 4.1, we initially set the increments for both the temperature and reward threshold to 0.1. To explore the effects of using finer-grained increments on B-STAR, we further conduct finer-grained hyper-parameters search with the granularity of 0.05 for temperature and 0.01 for reward threshold. Table 3 illustrates how B-STAR dynamically adjusts its configuration and the resulting impact on query effect. A comparison of Table 2 and Table 3 reveals that finer-grained configuration adjustments introduce more dynamic changes to temperature and reward thresholds throughout the training process, resulting in significantly higher query effect.

D Impact of Fixed Configuration Combinations

To confirm that the improvements achieved by B-STAR are due to its dynamic configuration adjustments rather than suboptimal configuration settings, we conduct a grid search to evaluate different configuration combinations for Online RFT. The temperature values are selected from the set [0.5,0.7,0.9,1.1] and the reward thresholds are chosen from [-0.4,-0.2,0.0,0.2,0.4]. For comparison, we also include two specific configurations: the default combination from our paper, (1.0,0.0), and the parameters obtained from B-STAR's final iteration, (1.1,-0.1).



(a) GSM 8K (b) MATH

Figure 7: Performance of Online RFT using different configuration combinations, where the horizontal axis represents changes in temperature and the vertical axis represents changes in reward threshold. B-STAR's GSM8K accuracy is 53.1%, while its MATH accuracy is 27.8%.

Configuration	**GSM 8K**	**MATH**
Temp = 1.0; Threshold = 0.0	46.8	23.2
Temp = 1.1; Threshold = -0.1	40.4	18.2
B-STaR	53.1	27.8

Table 4: Comparison of Online RFT using specific configurations and B-STaR Performance. This table reports the results with the stable hyperparameter combinations we found in our B-STaR experiments (Temperature = 1.1, Reward thresholds = -0.1) (Table 2).

Figure 7 and Table 4 illustrate that while grid search-based configuration combinations offer some performance improvements for online RFT, they remain less effective compared to the dynamic configuration adjustments enabled by B-STAR. This further emphasizes the critical need for dynamically balancing exploration and exploitation throughout the training process.

E Related Work of Dynamic Hyperparameter Adjustment

Dynamic hyperparameter optimization addresses the shortcomings of static configurations, which cannot adapt to the evolving dynamics of machine learning. Early work by Loshchilov & Hutter (2016) introduced the use of a cosine function to modulate learning rates, ensuring smoother convergence. Building on this, Baydin et al. (2017) proposed gradient-based methods to dynamically adjust learning rates in real-time by analyzing gradient trends, significantly accelerating convergence. Expanding the scope, Smith (2018) introduced a systematic approach to setting hyperparameters, such as learning rate, batch size, momentum, and weight decay, while highlighting their interdependence to improve training efficiency. Subsequently, Jaderberg et al. (2017) integrated model and hyperparameter optimization by asynchronously evolving a population of models through performance-based selection and mutation. Jomaa et al. (2019) framed hyperparameter tuning as a sequential decision-making problem, leveraging reinforcement learning to learn a policy for efficient hyperparameter selection. More recently, Baik et al. (2020) adopted a meta-learning framework to dynamically generate task- and step-specific hyperparameters, improving inner-loop optimization in few-shot learning tasks. Inspired by these innovations, our approach focuses on dynamically monitoring and balancing configurations between exploration and exploitation, optimizing the synergy between current policies and reward mechanisms to drive further performance gains.