Enhancing Reinforcement Learning Fine-Tuning with an Online Refiner

Anonymous ACL submission

Abstract

Constraints are essential for stabilizing reinforcement learning fine-tuning (RFT) and preventing degenerate outputs, yet they inherently conflict with the optimization objective because stronger constraints limit the ability of a fine-tuned model to discover better solutions. We propose *dynamic constraints* that resolve this tension by adapting to the evolving capabilities of the fine-tuned model based on the insight that constraints should only intervene when degenerate outputs occur. We implement this by using a reference model as an *online refiner* that takes the response from the fine-tuned model and generates a minimally corrected version which preserves correct content verbatim while fixing errors. A supervised fine-tuning loss then trains the fine-tuned model to produce the refined output. This mechanism yields a constraint that automatically strengthens or relaxes based on output quality. Experiments on dialogue and code generation show that dynamic constraints outperform both KL regularization and unconstrained baselines, achieving substantially higher task rewards while maintaining training stability.

1 Introduction

Reinforcement learning fine-tuning (RFT) has emerged as a powerful paradigm for aligning large language models (LLMs) with task-specific objectives, achieving remarkable success across diverse domains including safety alignment (Ji et al., 2023; Liu, 2023), code generation (Gu, 2023; Shojaee et al., 2023), and reasoning (Guo et al., 2025; Team et al., 2025). Recent algorithmic innovations, such as GRPO (Shao et al., 2024), DAPO (Yu et al., 2025), VAPO (Yuan et al., 2025), and GSPO (Zheng et al., 2025), have continually pushed the performance boundaries of RFT. However, as reinforcement learning (RL) plays an increasingly prominent role in post-training, it suffers from catastrophic forgetting (Chen et al., 2025; Korbak et al., 2022).

This phenomenon arises from the inherent difficulty of tuning a model in a high-dimensional exploration space with sparse rewards. The RL algorithm selectively reinforces task-specific knowledge, causing the policy distribution to become increasingly sharp and concentrated (Walder and Karkhanis, 2025). As the fine-tuned policy π_θ deviates substantially from the reference policy π_0, the general capabilities acquired during pretraining and supervised fine-tuning (SFT) are gradually eroded (Chen et al., 2025). In extreme cases, this catastrophic forgetting can severely impair the model's expressive capacity, leading to distribution collapse where the model degenerates into generating incoherent or nonsensical outputs (Korbak et al., 2022; Ma et al., 2024; Liu et al., 2025a; Mai et al., 2025).

A natural mitigation strategy is to impose a regularization term that constrains the fine-tuned policy from drifting too far from the reference one. KL divergence regularization, widely adopted in early RFT algorithms (Korbak et al., 2022; Padula and Soemers, 2024), serves precisely this purpose by anchoring π_θ to π_0. However, this approach introduces a fundamental trade-off. While it prevents catastrophic forgetting, it also limits π_θ to stay close to π_0, inherently restricting exploration and preventing the discovery of optimal policies that may reside in distant regions of the solution space (Yu et al., 2025).

Our key insight is that if we pursue extreme performance on specific tasks, we should abandon KL regularization, yet an alternative constraint remains essential to prevent the incoherent outputs induced by catastrophic forgetting. The fundamental limitation of KL regularization lies in its property of being a *static constraint*. In this paper, we propose

replacing this static constraint with a *dynamic constraint* that adapts to the evolving capabilities of the fine-tuned policy. Rather than anchoring to a fixed reference, the dynamic constraint guides π_θ toward an adaptive target that progresses alongside the policy.

We realize this idea by prompting the reference model π_0 to act as an *online refiner*. Given an input query and the response from π_θ, the refiner generates a refined version of the response. The dynamic constraint is then defined as the cross-entropy teaching π_θ to produce the refined response. This design maximally relaxes the constraint on π_θ by only intervening when degenerate outputs occur. When π_θ produces a coherent response, the refiner reproduces it verbatim and imposes almose no constraint. Only when degenerate outputs occur does the refiner apply targeted and minimal corrections. As illustrated in Figure 1, while the static constraint pulls the policy backward toward a fixed reference, the dynamic constraint propels it forward toward an adaptive, higher-quality target. From a broader perspective, our approach unifies RL with SFT on a continuously updated dataset.

We validate our approach through comprehensive experiments on dialogue and code generation tasks. We first integrate the dynamic constraint into PPO to analyze its training dynamics compared with static KL regularization, then incorporate it into DAPO (Yu et al., 2025), a state-of-the-art RFT algorithm, and evaluate on challenging code generation benchmarks. Our results reveal two compelling advantages. First, the dynamic constraint substantially stabilizes training by reducing variance and preventing distribution collapse. Second, it enables policies to achieve significantly higher rewards by eliminating the performance ceiling imposed by static constraints. These findings underscore the potential of dynamic constraints for advancing RFT. More broadly, our approach establishes a self-elevating cycle between RL and SFT, opening new avenues for improving both paradigms.

2 Preliminary

2.1 Problem Formulation

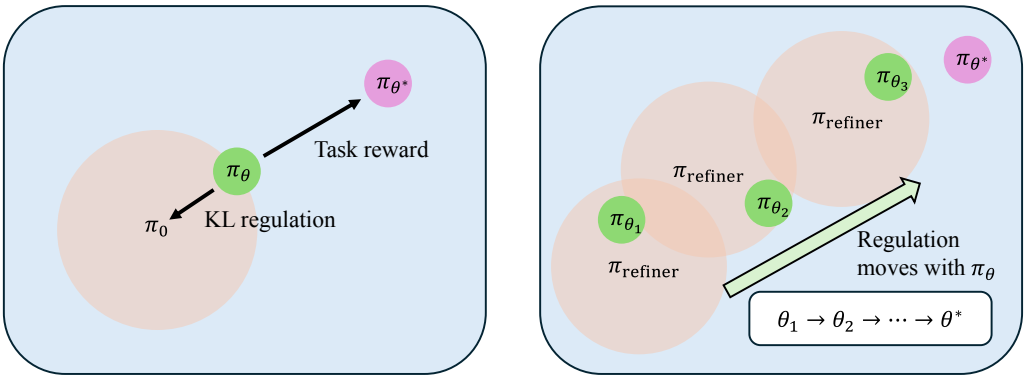
To describe RFT using standard RL terminology, we formulate next-token prediction in causal language models as a sequential decision-making process. This is formally modeled as a language-augmented Markov Decision Process (Li et al., 2022), defined as $\mathcal{M} = \langle \mathcal{V}, \mathcal{S}, \mathcal{A}, r, P, \gamma \rangle$. Here, $\mathcal{V}$ denotes the vocabulary of the language model, consisting of all possible tokens, and $w \in \mathcal{V}$ refers to a token. The state space is defined as $\mathcal{S} \subset \mathcal{V}^M$, where $\mathcal{V}^M$ represents the set of all token sequences of length up to M. Similarly, the action space is $\mathcal{A} \subset \mathcal{V}^N$, where $\mathcal{V}^N$ denotes token sequences of length up to N. Here, M and N are the maximum token lengths for states and actions, respectively. A state $s \in \mathcal{S}$ is represented as a token sequence $s = (w_1, w_2, \ldots, w_M)$, and an action $a \in \mathcal{A}$ is defined as a generated sequence $a = (w_1, w_2, \ldots, w_N)$. For simpliciy, we denote $(w_1, \ldots, w_i, \ldots, w_{t-1})$ as $a_{<t}$, and w_i as a_i. If the sequence length is shorter than the maximum, it is padded with a special token. The reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ assigns a task-specific scalar reward to each state-action pair. The transition function $P : \mathcal{S} \times \mathcal{V} \to \mathcal{S}$ defines a deterministic transition based on autoregressive generation. At each timestep, the predicted token is appended to the current state: $s_i = (s_{i-1}, a_i) = (s_0, a_{<i+1})$, where s_0 is the tokenized user input and $a_{<i+1}$ is the partial output sequence. The token-level policy of the language model is denoted by $\pi(a_i \mid s_0, a_{<i})$, and the sentence-level policy is defined as the product of token-level predictions: $\pi(a|s_0) = \prod_{i=1}^{N} \pi(a_i|s_0, a_{<i})$. The task reward $r(s_0, a)$ is only available after the full sequence a is generated.

2.2 Mirror Learning

Mirror learning (Grudzien et al., 2022) provides a unified theoretical framework for modern RL algorithms, including TRPO (Schulman et al., 2015) and PPO (Schulman et al., 2017). Policy optimization under mirror learning is expressed as:

$$\pi_{\text{new}} = \underset{\bar{\pi} \in \mathcal{N}(\pi_{\text{old}})}{\arg\max} \mathbb{E}_{s \sim \beta_{\pi_{\text{old}}}, a \sim \bar{\pi}} \Big[A_{\pi_{\text{old}}}(s, a) - \mathfrak{D}_{\pi_{\text{old}}}(\bar{\pi} \mid s) \Big], \tag{1}$$

where $\mathcal{N}(\pi_{\text{old}})$ denotes the neighborhood of π_{old}, and $\mathfrak{D}_{\pi_{\text{old}}}(\bar{\pi} \mid s)$ is a drift term that quantifies the divergence between π_{old} and a candidate policy $\bar{\pi}$. This framework guarantees convergence to the optimal policy if the operator $\mathcal{N}$ and $\mathfrak{D}$ satisfy mild requirements (see Appendix D for details). Modern RFT algorithms, such as PPO, GRPO, and DAPO, can all be viewed as instantiations of this general form, where the drift term controls update steps



(a) Static constraint (b) Dynamic constraint

Figure 1: **An illustration of the insight of dynamic constraint.** (a) The left figure illustrates the conventional KL regularization, which constrains the fine-tuned policy π_θ to remain close to the reference policy π_0. However, when the optimal policy lies far from π_0 and requires π_θ to deviate significantly, the KL regularization becomes an obstacle to policy optimization. (b) The right figure presents the insight of the dynamic constraint, where the constraint is derived from π_0's refined response based on π_θ's response and context. When π_θ deviates substantially from π_0, the dynamic constraint not only avoids hindering policy optimization but also provides effective guidance and correction for π_θ.

and the neighborhood defines the feasible search region.

3 Method

3.1 RFT under a Static Constraint

Current RL algorithms for fine-tuning LLMs typically include both task reward and KL regularization. The KL regularization is a constraint term that is transformed into an optimization objective. By treating the KL regularization as a constraint, the PPO algorithm can be viewed as solving the constrained optimization problem shown below.

$$\pi_{\text{new}} = \arg\max_{\bar{\pi}\in\mathcal{N}(\pi_0)} \mathbb{E}_{s_0\sim D,\{a_i\}_{i=1}^t\sim\pi_{\text{old}}} \left[\min\left(\mathrm{r}(\bar{\pi})A_{\pi_{\text{old}},t}, \mathrm{clip}(\mathrm{r}(\bar{\pi}),1\pm\epsilon)A_{\pi_{\text{old}},t}\right)\right], \tag{2}$$

$$\mathcal{N}(\pi_0) = \{\bar{\pi}\in\Pi \mid D_{\text{KL}}(\bar{\pi}\mid\pi_0)\le\delta\}, \tag{3}$$

where we denote $A_{\pi_{\text{old}}}(s_0, a_{<t}, a_t)$ as $A_{\pi_{\text{old}},t}$ for simplicity. The KL regularization constrains π_θ to remain close to the reference model π_0. This restriction can prevent π_θ from reaching the optimal policy π_{θ^*} and disrupts the convergence guarantee of standard PPO (Appendix D).

If the deviation between the fine-tuned model and the reference model is small, the KL regularization may have little influence on learning dynamics (Bai et al., 2022). However, when a larger deviation is desired, the KL regularization becomes a constraint that hinders progress. Since the KL divergence is measured relative to a fixed model, it restricts the direction of policy updates. This observation is consistent with recent work on reasoning models (Guo et al., 2025; Team et al., 2025; Abdin et al., 2025) which suggests that reasoning capabilities depend on starting from a strong model.

Despite its limitations, removing the KL regularization is often unwise. RL for LLMs operates in an extremely large action space and faces sparse rewards. Without any form of constraint, policy optimization often leads to incoherent or meaningless outputs. This issue is known as distribution collapse (Korbak et al., 2022; Ma et al., 2024).

These factors create the KL regularization challenge. The KL term helps prevent incoherent outputs but also limits how far the policy can move from the reference model. The main task is to allow the model to explore different behaviors without causing distribution collapse.

3.2 RFT under a Dynamic Constraint

To address this challenge, we propose a dynamic constraint that provides timely corrections while adapting as π_θ changes. This approach is motivated by the concept of a reference policy that stays near the current policy but avoids degenerate behaviors such as hallucinations. Since such a policy is not directly available, we approximate it by using the in-context learning capabilities of the reference model

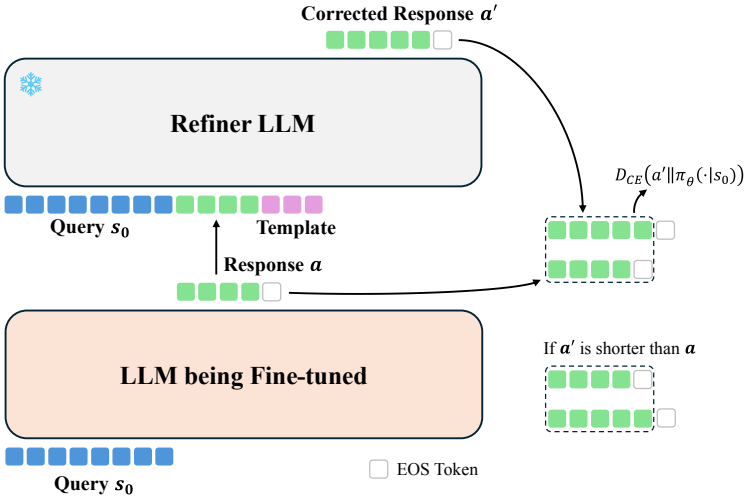


Figure 2: **The pipeline for calculating a dynamic constraint.** After π_θ generates a response a for the query s_0, a refiner LLM is employed to refine the response based on the necessary context (quesry s_0, response a) and a predefined template. The refinement process is conservative. In most cases, the refiner simply repeats a when no obvious issues are detected, while in other cases it makes only minimal edits to a when necessary. The detailed template is provided in Appendix A. The dynamic constraint is computed as a cross entropy loss that treats the refined response a' as the ground truth, which is equivalent to the SFT loss on the pair $\langle s_0, a' \rangle$.

π_0.

As illustrated in Figure 2, we first generate a response a from π_θ given query s_0. Then the query s_0 and response a are provided as input to the reference model π_0 to produce an improved response a'. This refined response serves as a dynamic anchor to guide the learning process. The dynamic constraint provides structure-preserving regularization without overly restricting policy improvement.

$$\pi_{\text{new}} = \arg\max_{\bar{\pi} \in \mathcal{N}(\pi_{\text{refiner}})} \mathbb{E}_{s_0 \sim D, \{a_i\}_{i=1}^{t} \sim \pi_{\text{old}}} \left[\min\left(\mathrm{r}(\bar{\pi}) A_{\pi_{\text{old}},t}, \mathrm{clip}(\mathrm{r}(\bar{\pi}), 1 \pm \epsilon) A_{\pi_{\text{old}},t} \right) \right], \tag{4}$$

$$\mathcal{N}(\pi_{\text{refiner}}) = \{ \bar{\pi} \in \Pi \mid D(\bar{\pi} \mid \pi_{\text{refiner}}) \leq \delta \}. \tag{5}$$

In this formulation, π_{refiner} denotes the refiner output $\pi_0(\cdot \mid s_0, a)$ where a is sampled from π_θ. Although the refiner can correct errors while preserving correct parts of the output, we notice that for the preserved tokens, π_{refiner} can at best ensure that $arg\max\ \pi_0(\cdot|s_i, a) = arg\max\ \pi_\theta(\cdot|s_i)$, but it does not guarantee that $\pi_0(\cdot|s_i, a) = \pi_\theta(\cdot|s_i)$. So, directly applying a KL divergence constraint of the form $D_{\text{KL}}(\pi_0(\cdot \mid s_i, a), \bar{\pi}(\cdot \mid s_i))$ is not appropriate. This is because $\pi_0(\cdot \mid s_i, a)$ is only guaranteed to be more reliable than $\bar{\pi}(\cdot \mid s_i)$ at the token level, and does not necessarily represent a better distribution overall.

3.3 The Implementation of Dynamic Constraint

We define the dynamic constraint in the form of a sequence-level cross entropy under self-feeding:

$$D(\bar{\pi}|\pi_{\text{refiner}}) = \sum_{t=1}^{\min\{|a|,|a'|\}} -\log\left(\pi_\theta(a'_t \mid s_0, a'_{<t})\right), \quad \text{s.t. } r(s_0, a') > r(s_0, a), \tag{6}$$

where a is the response sampled from π_θ given s_0, a' is the refined response conditioned on (s_0, a), and $r(\cdot, \cdot)$ denotes the reward function. In practice, we implement the dynamic constraint as an additive penalty: at step t, we add $-\eta \log\left(\pi_\theta(a'_t \mid s_0, a'_{<t})\right)$ on the reward, where η is a penalty weight. This effectively integrates an SFT-style loss into the RL objective. The refined answer a', paired with the question s_0, forms an SFT training example $\langle s_0, a' \rangle$. Unlike prior RFT–SFT mixed training methods (Lv et al., 2025; Liu et al., 2025c; Huang et al., 2025), our SFT data come from a dynamic dataset (Fig. 3), in which a' is distributed around a. This prevents π_θ from being confined to the neighborhood of a fixed reference policy.

Ideally, since the refiner can always choose to repeat a and only refine when a is problematic, the refined output a' should never be worse than a. In practice, however, the refiner is imperfect,

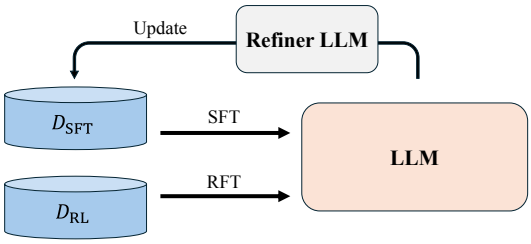


Figure 3: **Dynamic constraint from a dataset perspective.** The dynamic constraint can be interpreted as an RFT-SFT hybrid training approach with a dynamically updated SFT dataset.

and cases where a' is worse than a are unavoidable. Moreover, as the capability of π_θ improves during training, the proportion of such worse a' tends to increase. To prevent these inferior a' from negatively affecting the training of π_θ, we filter them out and exclude them from providing constraints.

4 Experiment

Our experimental evaluation is designed to answer two fundamental questions: (1) **Dynamics**: Can dynamic constraints effectively resolve the tension between training stability and unbounded exploration? (2) **Performance**: Does this mechanism yield state-of-the-art results on complex tasks compared to strong RFT baselines?

4.1 Analysis on Training Dynamics

Setup. We investigate whether dynamic constraints can resolve the fundamental trade-off between training stability and exploration capability. We compare our *Dynamic* approach against two key baselines: *Static*, which employs standard KL regularization anchored to a fixed pretrained model, and *w/o Constraint*, which performs pure RL optimization without any regularization. All constraint variants are implemented using PPO as the underlying RL algorithm.

The experiments span two benchmarks, including Prompt-Collection-v0.1 (Team, 2024), and APPS (Hendrycks et al., 2021). The Prompt-Collection-v0.1 dataset consists of 179,000 QA samples for training, including prompts from six subsets: UltraFeedback (Cui et al., 2024), HelpSteer (Wang et al., 2024), OpenOrca Pairs (Lian et al., 2023), UltraInteract (Yuan et al., 2024), DIBT 10K Prompts Ranked, and Capybara Preferences (Argilla, 2024). We employ a reward model[1] to provide the rewards. The APPS dataset comprises 10,000 problems sourced from open-access coding platforms such as Codeforces, Kattis, and others, with 5,000 problems for training and 5,000 for testing. This dataset is designed to evaluate the ability of language models to generate code from natural language specifications. Our reward model follows the framework of (Le et al., 2022) and (Liu et al., 2023), utilizing a verifiable reward function based on compile and unit test results.

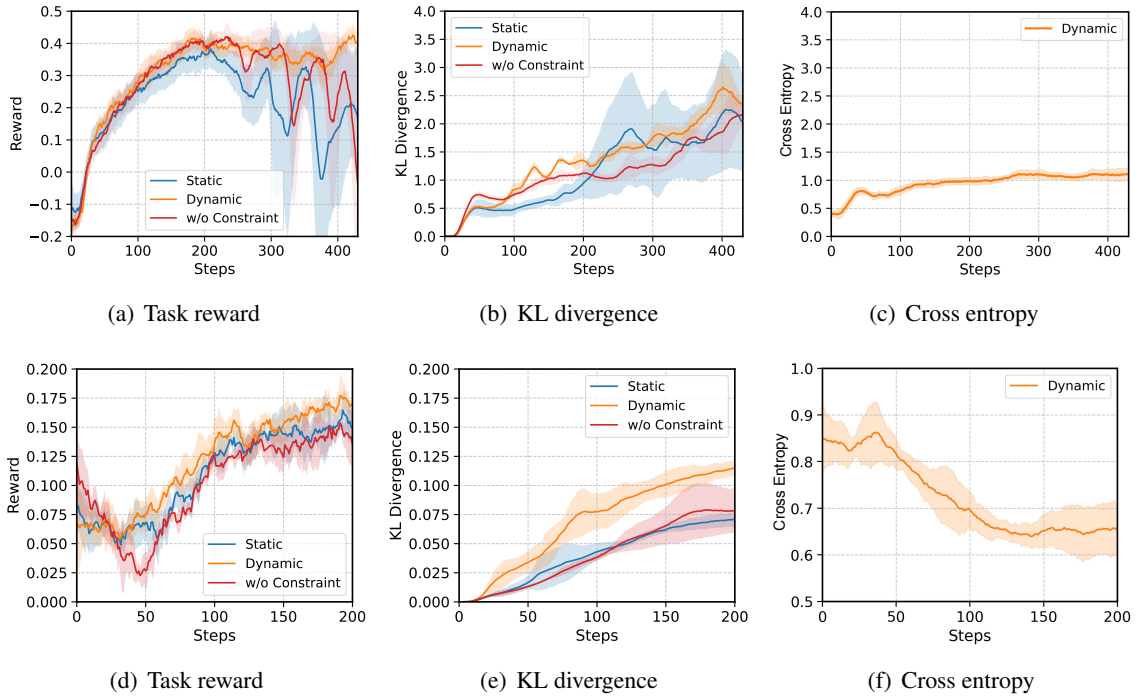
We employ Llama-3.2-3B-Instruct[2] as our base model for training on both Prompt-Collection-v0.1 and APPS datasets. For comprehensive analysis, we report training curves for task rewards and KL divergence between the reference policy π_0 and the current policy π_θ across all benchmarks. Note that while the *Dynamic* method does not explicitly use KL divergence as a constraint, we compute it for comparative analysis alongside the *w/o Constraint* baseline. Both *Dynamic* and *Static* methods use Llama-3.2-3B-Instruct as their reference model.

Reward and KL Dynamics. Figure 4 reveals a fundamental difference in optimization trajectories. The reward curve for the Dynamic method increases steadily alongside a monotonically increasing KL divergence. This confirms that π_θ continuously explores new regions of the parameter space, unhindered by a fixed anchor. In contrast, the Static baseline sees its KL divergence plateau early, suggesting the policy has hit the boundary of the trust region defined by π_0, effectively halting further exploration. The w/o Constraint baseline initially improves but eventually suffers from distribution collapse, evidenced by a sudden drop in reward and a spike in KL divergence. This result validates our hypothesis that dynamic constraints enable sustained exploration where static constraints saturate and unconstrained RL collapses.

Online Refiner Stability. The stability of the cross-entropy loss serves as a proxy metric for the refiner's reliability. Despite the policy π_θ drifting significantly from its initialization (high KL), the distance between π_θ and π_{refiner} (cross entropy) remains low and stable. This implies that the refiner successfully acts as a "moving anchor," adapting to the policy's evolution while maintaining local regularization. The refiner effectively approximates a gold model, guiding the policy through the optimization landscape without tethering it to the start-

[1] https://huggingface.co/OpenRLHF/Llama-3-8b-rm-mixture

[2] https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct



(a) Task reward (b) KL divergence (c) Cross entropy

(d) Task reward (e) KL divergence (f) Cross entropy

Figure 4: **Training dynamics on Prompt-Collection-v0.1 (top) and APPS (bottom).** (a, d) *Dynamic* (orange) achieves continuous reward growth, whereas *Static* (blue) saturates and *w/o constraint* (red) collapses. (b, e) The KL divergence of *Dynamic* rises steadily, indicating deep exploration beyond the initial policy π_0, while *Static* remains tethered. (c, f) The cross entropy remains low, confirming that the Refiner π_{refiner} successfully tracks the evolving policy π_θ.

ing point.

4.2 Comparing to DAPO on Code Generation

Setup. The analysis of training dynamics mainly serves to understand the mechanism of dynamic constraint. To further compare the practical performance of our method against state-of-the-art approach, we select DAPO as the baseline, and implement the dynamic constraint based on DAPO by using it as a penalty like Ouyang et al. (2022). Implementation details for both methods are provided in Appendix A.1.

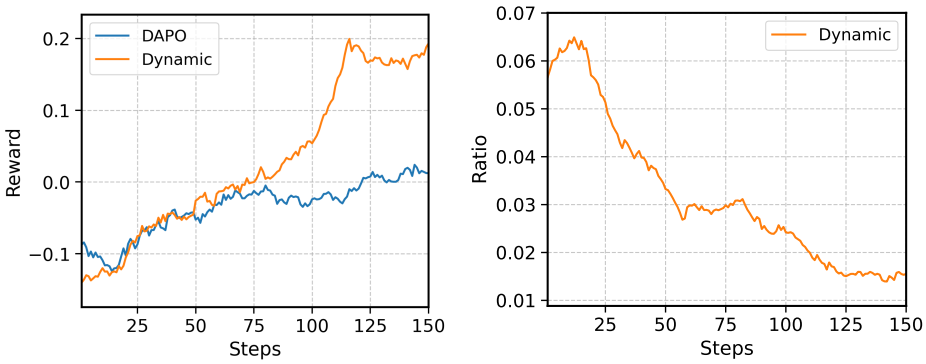
We employ RFT on the training set of APPS, and use the same verifiable reward function as in Sec. 4.1. After training, we evaluate both methods on four Python code-generation benchmarks: HumanEval, HumanEval+, MBPP, and MBPP++ and report PASS@1 using bigcode-evaluation-harness[3]. In practice, RFT for code generation typically begins with a strong checkpoint to maximize performance gains. Accordingly, we adopt Qwen2.5-Coder-1.5B-Instruct[4], a model trained on large-scale dataset that includes extensive code-centric corpora. All methods perform fine-tuning based on Qwen2.5-Coder-1.5B-Instruct.

Benchmarking Results. As shown in Figure 5(a), *Dynamic* achieves a significantly higher asymptotic reward than DAPO. Notably, the reward curve exhibits a sharp acceleration around step 75, suggesting the discovery of novel optimization pathways that were likely suppressed by static regularization. Table 1 confirms that this advantage generalizes. *Dynamic* outperforms DAPO by a significant margin (+30.8% relative gain on average) across all test sets. This result suggests that dynamic constraints not only improve in-domain optimization but also preserve general coding capabilities better than static regularization.

Adaptive Intervention. Figure 5(b) plots the "improvement ratio" indicating the fraction of rollouts where the Refiner yields a higher reward than the policy π_θ. This ratio naturally decays as the policy improves, confirming that our constraint is self-annealing. The dynamic constraint provides strong guidance early in training and relaxes as the policy becomes competent. Nevertheless, even

[3] https://github.com/bigcode-project/bigcode-evaluation-harness

[4] https://huggingface.co/Qwen/Qwen2.5-Coder-1.5B-Instruct

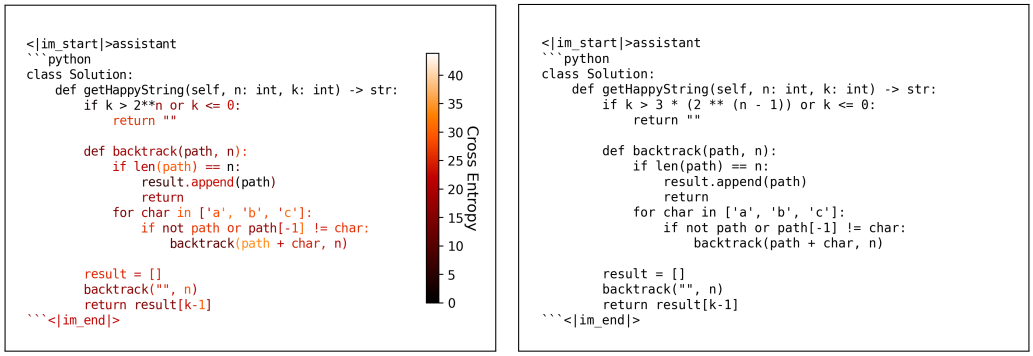


(a) Task reward (b) Improvement ratio of Refiner LLM

Figure 5: **Training curves compared with DAPO on the APPS dataset.** The right figure shows the proportion of rollouts improved by the Refiner LLM.

Model	HumanEval	HumanEval+	MBPP	MBPP+	Avg.
Qwen2.5-Coder-1.5B-Instruct	12.8	11.6	36.2	40.2	25.2
DAPO	18.0 (+40.6%)	15.9 (+37.1%)	39.4 (+8.8%)	43.9 (+9.2%)	29.3 (+16.3%)
Dynamic	20.7 (+61.7%)	19.9 (+71.6%)	44.4 (+22.7%)	46.8 (+16.4%)	33.0 (+30.8%)

Table 1: **Pass@1 (%) on Python code generation benchmarks.** Qwen2.5-Coder-1.5B-Instruct is fine-tuned using DAPO and our method, respectively, and evaluated on four code generation benchmarks with the `bigcode-evaluation-harness`.



(a) Original response (b) Refined response

Figure 6: **Demonstration of how the dynamic constraint refines model responses.** The two panels show Python backtracking solutions for the LeetCode problem "1415. The k-th Lexicographical String of All Happy Strings of Length n". (a) The left panel presents the original response, which contains an indexing error. (b) The right panel shows the refined response produced by the Refiner LLM, where the model enforces a more accurate condition (`3*(2*(n-1))`). The color on the left indicates the cross entropy values for each token.

when fewer than 10% of rollouts are improved, the dynamic constraint still provides effective optimization guidance. Further ablation studies on the filter mechanism and dynamic constraint coefficient η could be found in Appendix B.

Qualitative Analysis. Figure 6 visualizes the correction mechanism on a backtracking problem. The heatmap shows token-level cross-entropy. We observe that the constraint is sparse: for correct segments, the loss is near zero, activating only when the policy deviates into error. The Refiner successfully corrects a logical indexing error, guiding the policy from a failing state ($r = -0.3$) to a perfect solution ($r = 1.0$). This "intervention-on-demand" behavior contrasts sharply with static KL, which indiscriminately penalizes deviation. Additional examples are provided in Appendix C.

Computational Cost Analysis. The introduction of the online refiner tends to result in additional inference overhead. We compare the training

time of DAPO and our Dynamic method using 4 NVIDIA A6000 GPUs. The total training time for DAPO is approximately 17.8 hours, while the Dynamic method requires 26.3 hours. This corresponds to a 48% increase in training time. However, given the substantial performance improvement observed in benchmarks, we consider this trade-off to be reasonable in many cases.

Qualitative Analysis. Figure 6 presents a detailed example illustrating the effect of the dynamic constraint. We extract an original response and its corresponding refined response during training, and compute the token-wise cross entropy to analyze the constraint at the token level. The original response attains a reward of -0.3 (compile passes but unit test fails), whereas the refined response achieves a reward of 1.0, successfully correcting the errors in the original response. The refined response repeats the original content in the initial segments, resulting in overlapping tokens with cross entropy values close to zero. After the point of modification, the cross entropy rises sharply and remains high thereafter. The dynamic constraint applies to the entire sequence following the modification, since in a causal language model, changing a single token affects all subsequent tokens. Therefore, aligning the entire remaining sequence to the refined response is reasonable. Additional examples are provided in Appendix C.

5 Related Work

KL Regularization in RFT. The earliest work by Stiennon et al. (2020) introduces KL regularization to constrain the magnitude of policy updates during fine-tuning. This approach addresses the out-of-distribution (OOD) issue that arises because reward models are typically trained on outputs generated by the reference model and annotated by humans. InstructGPT (Ouyang et al., 2022) adopts the same mechanism for similar reasons, and subsequent models such as Kimi K1.5 (Team et al., 2025) and DeepSeek-R1 (Guo et al., 2025) continue this practice for training reasoning models. Following DAPO (Yu et al., 2025), KL regularization has been gradually removed from RFT algorithms as it restricts model performance. This aligns with the finding in Bai et al. (2022) that $\sqrt{D_{\mathrm{KL}}(\pi_\theta \| \pi_0)}$ correlates approximately linearly with task reward. However, ProRL (Liu et al., 2025b) demonstrates that KL regularization can help balance exploration and exploitation in the policy. Our work seeks to relax the KL divergence constraint to the greatest extent possible, striking a balance that preserves the stability it provides while preventing it from stifling the model's exploratory potential.

Hybrid RFT and SFT. To bridge the gap between the imitation limits of SFT (Shenfeld et al., 2025; Kirk et al., 2023) and the inherent instability of RFT (Lv et al., 2025), hybrid approaches like UFT (Liu et al., 2025c) and SRFT (Fu et al., 2025) jointly optimize these objectives to balance imitation and exploration. Recent work further integrates them via prefix-conditioning (Huang et al., 2025) or unified theoretical frameworks (Lv et al., 2025). However, these methods typically derive the supervised signal from static offline datasets, causing the SFT loss to act as a fixed regularization term that pulls the policy toward potentially suboptimal demonstrations. Our approach departs from this by introducing a dynamic constraint that adaptively refines policy rollouts to provide an evolving supervised signal. This online mechanism stabilizes training without tethering the policy to an outdated reference, thereby preserving exploration efficiency while constraining unsafe drift.

6 Conclusion

The fundamental limitation of KL regularization lies in its static nature. By anchoring the policy to a fixed reference, static constraints create a trade-off between stability and optimality. We address this by introducing dynamic constraints that evolve with the policy, transforming the reference from a fixed anchor into an adaptive guide.

Our key insight is that constraints should intervene only when necessary. By using the reference model as an online refiner that provides minimal corrections, the constraint automatically strengthens when outputs degrade and relaxes when they improve. This enables policies to explore beyond the pretrained distribution while maintaining language coherence.

Empirical results across dialogue and code generation tasks demonstrate that dynamic constraints resolve the KL regularization dilemma. Policies achieve substantially higher rewards while maintaining stability, even as they diverge significantly from the reference model. Our work suggests that the future of reinforcement learning fine-tuning may lie not in removing constraints, but in making them adaptive.

References

Marah Abdin, Sahaj Agarwal, Ahmed Awadallah, Vidhisha Balachandran, Harkirat Behl, Lingjiao Chen, Gustavo de Rosa, Suriya Gunasekar, Mojan Javaheripi, Neel Joshi, and 1 others. 2025. Phi-4-reasoning technical report. *arXiv preprint arXiv:2504.21318*.

Argilla. 2024. Capybara-preferences dataset.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, and 1 others. 2022. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*.

Liang Chen, Xueting Han, Li Shen, Jing Bai, and Kam-Fai Wong. 2025. Beyond two-stage training: Cooperative sft and rl for llm reasoning. *arXiv preprint arXiv:2509.06948*.

Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Bingxiang He, Wei Zhu, Yuan Ni, Guotong Xie, Ruobing Xie, Yankai Lin, Zhiyuan Liu, and Maosong Sun. 2024. ULTRAFEEDBACK: Boosting language models with scaled AI feedback. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 9722–9744. PMLR.

Yuqian Fu, Tinghong Chen, Jiajun Chai, Xihuai Wang, Songjun Tu, Guojun Yin, Wei Lin, Qichao Zhang, Yuanheng Zhu, and Dongbin Zhao. 2025. Srft: A single-stage method with supervised and reinforcement fine-tuning for reasoning. *arXiv preprint arXiv:2506.19767*.

Jakub Grudzien, Christian A Schroeder De Witt, and Jakob Foerster. 2022. Mirror learning: A unifying framework of policy optimisation. In *International Conference on Machine Learning*, pages 7825–7844. PMLR.

Qiuhan Gu. 2023. Llm-based code generation method for golang compiler testing. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 2201–2203.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. 2021. Measuring coding challenge competence with apps. *NeurIPS*.

Zeyu Huang, Tianhao Cheng, Zihan Qiu, Zili Wang, Yinghui Xu, Edoardo M. Ponti, and Ivan Titov. 2025. Blending supervised and reinforcement fine-tuning with prefix sampling. *arXiv preprint arXiv:2507.01679*.

Jiaming Ji, Mickel Liu, Josef Dai, Xuehai Pan, Chi Zhang, Ce Bian, Boyuan Chen, Ruiyang Sun, Yizhou Wang, and Yaodong Yang. 2023. Beavertails: Towards improved safety alignment of llm via a human-preference dataset. *Advances in Neural Information Processing Systems*, 36:24678–24704.

Robert Kirk, Ishita Mediratta, Christoforos Nalmpantis, Jelena Luketina, Eric Hambro, Edward Grefenstette, and Roberta Raileanu. 2023. Understanding the effects of rlhf on llm generalisation and diversity. *arXiv preprint arXiv:2310.06452*.

Tomasz Korbak, Ethan Perez, and Christopher L Buckley. 2022. Rl with kl penalties is better viewed as bayesian inference. *arXiv preprint arXiv:2205.11275*.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven Chu Hong Hoi. 2022. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. *Advances in Neural Information Processing Systems*, 35:21314–21328.

Shuang Li, Xavier Puig, Chris Paxton, Yilun Du, Clinton Wang, Linxi Fan, Tao Chen, De-An Huang, Ekin Akyürek, Anima Anandkumar, and 1 others. 2022. Pre-trained language models for interactive decision-making. *Advances in Neural Information Processing Systems*, 35:31199–31212.

Wing Lian, Bleys Goodson, Eugene Pentland, Austin Cook, Chanvichet Vong, and "Teknium". 2023. Openorca: An open dataset of gpt augmented flan reasoning traces. https://huggingface.co/Open-Orca.

Jiate Liu, Yiqin Zhu, Kaiwen Xiao, QIANG FU, Xiao Han, Yang Wei, and Deheng Ye. 2023. Rltf: Reinforcement learning from unit test feedback. *Transactions on Machine Learning Research*.

Mingjie Liu, Shizhe Diao, Jian Hu, Ximing Lu, Xin Dong, Hao Zhang, Alexander Bukharin, Shaokun Zhang, Jiaqi Zeng, Makesh Narsimhan Sreedhar, and 1 others. 2025a. Scaling up rl: Unlocking diverse reasoning in llms via prolonged training. *arXiv preprint arXiv:2507.12507*.

Mingjie Liu, Shizhe Diao, Ximing Lu, Jian Hu, Xin Dong, Yejin Choi, Jan Kautz, and Yi Dong. 2025b. Prorl: Prolonged reinforcement learning expands reasoning boundaries in large language models. *arXiv preprint arXiv:2505.24864*.

Mingyang Liu, Gabriele Farina, and Asuman Ozdaglar. 2025c. Uft: Unifying supervised and reinforcement fine-tuning. *arXiv preprint arXiv:2505.16984*.

Yang Liu. 2023. The importance of human-labeled data in the era of llms. *arXiv preprint arXiv:2306.14910*.

Xingtai Lv, Yuxin Zuo, Youbang Sun, Hongyi Liu, Yuntian Wei, Zhekai Chen, Lixuan He, Xuekai Zhu, Kaiyan Zhang, Bingning Wang, Ning Ding, and Bowen Zhou. 2025. Towards a unified view of large language model post-training. *arXiv preprint arXiv:2509.04419*.

Hao Ma, Tianyi Hu, Zhiqiang Pu, Liu Boyin, Xiaolin Ai, Yanyan Liang, and Min Chen. 2024. Coevolving with the other you: Fine-tuning llm with sequential cooperative multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 37:15497–15525.

Xinji Mai, Haotian Xu, Weinong Wang, Jian Hu, Yingying Zhang, Wenqiang Zhang, and 1 others. 2025. Agent rl scaling law: Agent rl with spontaneous code execution for mathematical problem solving. *arXiv preprint arXiv:2505.07773*.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, and 1 others. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Alexander G Padula and Dennis JNJ Soemers. 2024. Exploring rl-based llm training for formal language tasks with programmed rewards. *arXiv preprint arXiv:2410.17126*.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Idan Shenfeld, Jyothish Pari, and Pulkit Agrawal. 2025. Rl's razor: Why online reinforcement learning forgets less. *arXiv preprint arXiv:2509.04259*.

Parshin Shojaee, Aneesh Jain, Sindhu Tipirneni, and Chandan K Reddy. 2023. Execution-based code generation using deep reinforcement learning. *arXiv preprint arXiv:2301.13816*.

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. 2020. Learning to summarize with human feedback. *Advances in neural information processing systems*, 33:3008–3021.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, and 1 others. 2025. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*.

OpenRLHF Team. 2024. Openrlhf/prompt-collection-v0.1. https://huggingface.co/datasets/OpenRLHF/prompt-collection-v0.1.

Christian Walder and Deep Tejas Karkhanis. 2025. Pass@k policy optimization: Solving harder reinforcement learning problems. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Zhilin Wang, Yi Dong, Jiaqi Zeng, Virginia Adams, Makesh Narsimhan Sreedhar, Daniel Egert, Olivier Delalleau, Jane Scowcroft, Neel Kant, Aidan Swope, and Oleksii Kuchaiev. 2024. Helpsteer: Multi-attribute helpfulness dataset for steerlm. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3371–3384. Association for Computational Linguistics.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, and 1 others. 2025. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*.

Lifan Yuan, Ganqu Cui, Hanbin Wang, Ning Ding, Xingyao Wang, Jia Deng, Boji Shan, Huimin Chen, Ruobing Xie, Yankai Lin, Zhenghao Liu, Bowen Zhou, Hao Peng, Zhiyuan Liu, and Maosong Sun. 2024. Advancing llm reasoning generalists with preference trees. *Preprint*, arXiv:2404.02078.

Yufeng Yuan, Qiying Yu, Xiaochen Zuo, Ruofei Zhu, Wenyuan Xu, Jiaze Chen, Chengyi Wang, TianTian Fan, Zhengyin Du, Xiangpeng Wei, and 1 others. 2025. Vapo: Efficient and reliable reinforcement learning for advanced reasoning tasks. *arXiv preprint arXiv:2504.05118*.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, and 1 others. 2025. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*.

A Experiments Details

A.1 Hyperparameters

We summarize the hyperparameter settings used in Table 2. The exmperiment codes are based on `OpenRLHF`[5]. Most hyperparameters follow the default configurations provided by the respective codebases. We adjust the batch size according to available computational resources, and focus primarily on tuning the learning rate and KL coefficient.

We adopt a sequential tuning strategy: we first search for an appropriate learning rate, then tune the KL coefficient. Intentionally, we increase the learning rate to the point where, under a default KL coefficient, the PPO training with a static constraint begins to exhibit distributional collapse. This setup is designed to satisfy our core hypothesis (illustrated in Figure 1): once π_θ moves beyond the "safe region" around π_0, the static constraint hinders further updates, while the dynamic constraint does not. After identifying this collapse point, we gradually refine the KL coefficient until the static constraint is able to maintain a balance between stable training and performance. Then, the same hyperparameters are directly applied on the dynamic constraint.

Our experimentation employs 2 AMD EPYC 7773X CPUs and 8 NVIDIA A6000 GPUs (48GB each).

Table 2: Hyperparameters in Section 4.1.

Hyperparameter	**Prompt-Collection-v0.1**	**APPS**
Actor Learning Rate	1e-6	1e-6
Critic Learning Rate	9e-6	9e-6
Epochs	1	7
PPO Epoch	1	4
Batch Size	64	128
Mini Batch Size	16	16
Gradient Accum. Steps	4	8
Iterations	430	200
KL Coefficient (η)	1e-2	1e-4
Discount (γ)	1	1
GAE (λ)	0.95	0.95
Clip Ratio (ϵ)	0.2	0.2
Value Clip Range	0.2	0.2

A.2 Reward Setting for Code Generation

We implement a two-tiered reward mechanism combining coarse-grained feedback and adaptive testing incentives. The coarse-grained reward follows the same setting as (Le et al., 2022) :

$$R_{\text{coarse}}(\hat{W}) = \begin{cases} -1.0, & FB(\hat{W}) \text{ is syntax error} \\ -0.6, & FB(\hat{W}) \text{ is runtime error} \\ -0.3, & FB(\hat{W}) \text{ is unit test failure} \\ 1.0, & FB(\hat{W}) \text{ is all pass} \end{cases} \tag{7}$$

Coarse-grained feedback serves as an incentive mechanism for language models, increasing the probability of generating correcte code reduceing the chances of producing erroneous code. When the $FB(\hat{W})$ is between pass and failure, we introduce adaptive feedback proportional to the test passing rate to address reward sparsity and encour-

Table 3: Hyperparameters in Section 4.2.

Hyperparameter	**DAPO**	**Dynamic**
Actor Learning Rate	1e-6	1e-6
Epochs	5	5
Batch Size	64	64
Mini Batch Size	8	8
Samples per Prompt	8	8
Iterations	150	150
KL Coefficient (η)	1e-2	1e-4
Discount (γ)	1	1
Clip Ratio ($\epsilon_{\text{low}}, \epsilon_{\text{high}}$)	(0.2, 0.3)	(0.2, 0.3)
Value Clip Range	0.5	0.5

[5] https://github.com/OpenRLHF/OpenRLHF

age maximal test coverage.

$$R_{\text{adaptive}}(\hat{W}) = -0.3 + 1.3 \times \frac{N_{\text{pass}}}{N_{\text{pass}} + N_{\text{fail}}} \quad (8)$$

Where $FB(\hat{W})$ represents the compiler feedback for generated code $\hat{W}$,and the value of $R_{\text{adaptive}}(\hat{W})$ is determined by the pass rate of the unit test.

A.3 Prompt Detail

The following prompt is used to instruct the Refiner LLM in the APPS code generation experiments. The prompt guides the refiner to either reproduce correct solutions verbatim or apply minimal corrections to incorrect ones.

Refiner Prompt for APPS

You are a coding assistant for refining program solutions. You will receive:
- Question: A programming problem description.
- Response: A proposed code solution.
- Reward: A numerical score measuring correctness, defined as:
- -1.0 → Compilation failed
- -0.6 → Runtime error on test cases
- $-0.3 + 1.3 \times$ pass_rate → Partial pass, where pass_rate $\in [0, 1]$

Your Task:
1. If Reward = 1.0 (full correctness): Output the code from Response exactly, without changes.
2. If Reward < 1.0 (incorrect/incomplete):
- Analyze Response to identify issues (syntax errors, runtime errors, logical mistakes, failing test cases).
- Refine and fix the code to maximize correctness and improve reward.
- Ensure the refined code follows the input/output format, covers edge cases, and is efficient enough.
3. Output Format: Only output the final code (either the same code if correct, or a refined version). Do not include additional explanations, comments, or extra text.

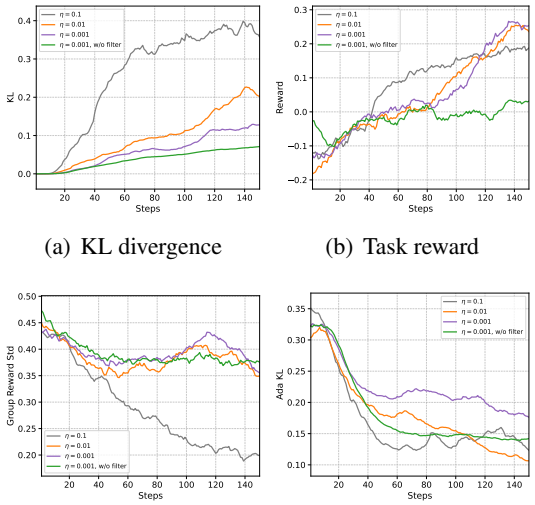
Question: {question}
Response: {response}
Reward: {reward}
Refine:

B Ablation Studies

We conduct ablation experiments to evaluate the effect of (1) the filtering mechanism described in Equation 6, which ensures that only refined responses with higher rewards than the original responses are used to provide dynamic constraints, and (2) the dynamic constraint coefficient η, which controls the strength of the cross-entropy regularization term.

Effect of η. As shown in Figure 7, a larger η leads to rapid reward improvement in the early stages of training. However, over time, it negatively impacts the diversity of generated outputs, as evidenced by the declining group reward standard deviation. This reduced diversity ultimately limits the final achievable reward. Based on these observations, we select $\eta = 0.001$ as a balanced choice that maintains sufficient exploration while still benefiting from the dynamic constraint.

Effect of Filter. Without the filtering mechanism, the refiner occasionally produces responses with lower rewards than the original outputs from π_θ. This phenomenon becomes more pronounced as π_θ improves during training, since stronger policies are harder to refine. Consequently, including such inferior refined responses degrades the quality of the dynamic constraint and hampers policy optimization. The filter effectively addresses this issue by excluding refinements that fail to improve upon the original response.



(a) KL divergence (b) Task reward

(c) Group reward std (d) Cross entropy

Figure 7: **Ablation studies on the filtering mechanism and dynamic constraint coefficient η.** The curves show training dynamics under different configurations: with/without the filter that excludes refined responses with lower rewards, and with different values of η that control the strength of the dynamic constraint.

C Qualitative Analysis

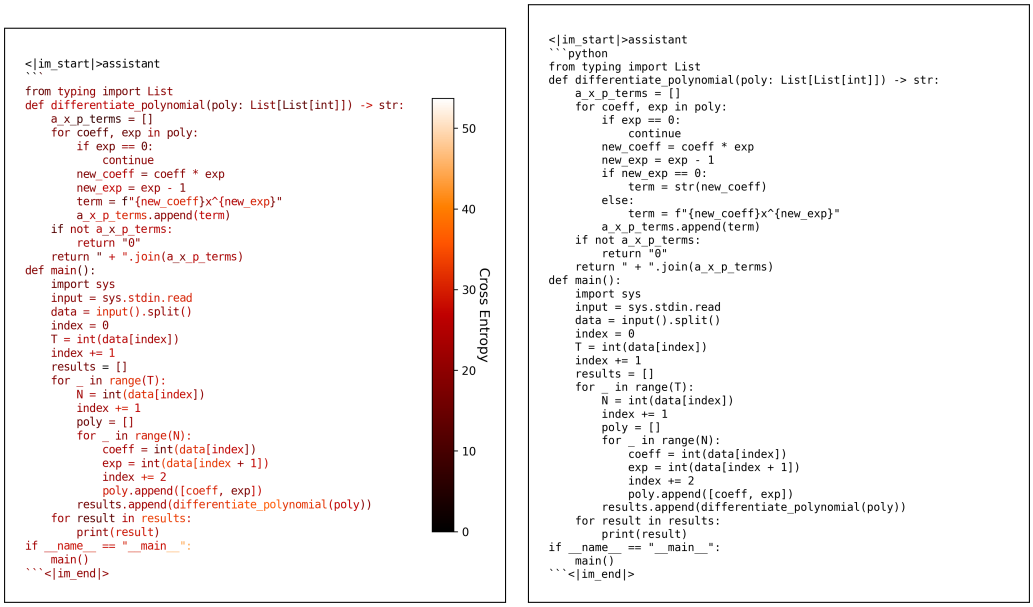
We provide a qualitative analysis on the `Prompt-Collection-v0.1` dataset in Table 4. The first two examples show cases where π_θ produces accurate and appropriate answers. In these cases, π_{refiner} stays close to π_θ, while π_0 can drift and yield noticeably different outputs. The third example shows a truncation failure where π_θ hits the

Table 4: Examples of Prompt-Collection-v0.1.

Query s_0	**Responses / Model Outputs**
Name a kind of fungal hyphae that can live in outer space.	**Response of π_θ:** Armillaria ostoyae, a type of honey mushroom, has been found in space debris.
	Response of π_0: Armillaria mellea.
	Response of π_{refiner}: Armillaria ostoyae, a type of honey mushroom, has been found in space debris.
Compute this: 7299729 + 114. Exclude words; show only the math.	**Response of π_θ:** 7299729 + 114 = 7299843
	Response of π_0: 7299729 + 114 = 7299843
	Response of π_{refiner}: 7299729 + 114 = 7299843
What is the history of the Oxford dictionary? How did the dictionary come to be?	**Response of π_θ:** Oxford Dictionary originated in 1884, initially a supplement to the Oxford English Dictionary (OED). First published 1884, updated by contributors, including...
	Response of π_0: Founded in 1884 by James Murray, it evolved from a 14-volume set of dictionaries compiled by Samuel Johnson and others.
	Response of π_{refiner}: Oxford Dictionary originated in 1884, initially a supplement to the Oxford English Dictionary (OED).
Who is faster in real life, a tortoise or a hare?	**Response of π_θ:** The tortoise.
	Response of π_0: The tortoise.
	Response of π_{refiner}: The hare.

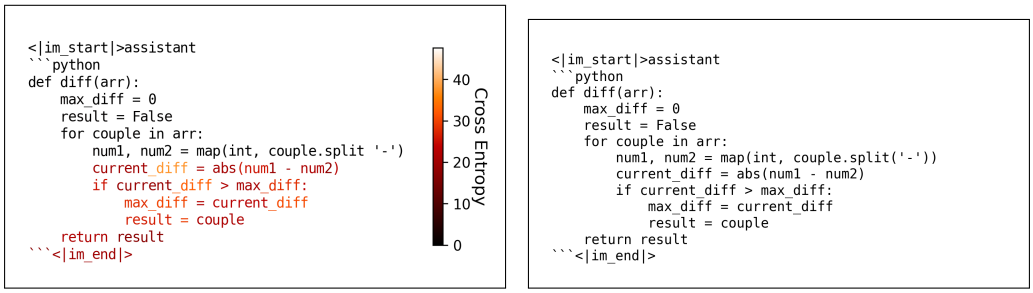
maximum generation length and ends with an incomplete sentence. π_{refiner} detects this issue and keeps only the first complete sentence, removing the broken fragment. This example also highlights that when π_0 is far from π_θ, it can act as a strong constraint even when π_θ is semantically correct. In the final example, both π_θ and π_0 give an incorrect answer, which may reflect biased knowledge from pretraining such as the tortoise and hare story. In contrast, π_{refiner} corrects the error and returns the correct answer.

We also include qualitative examples on the APPS dataset in Figures 8, 9, 10, and 11. Across these four cases, refinement consistently improves the reward from a negative value to 1.0. The refiner restores missing format signals such as the `python` header, removes redundant operations, and makes the logic easier to follow. It also fixes wrong computation branches by rewriting conditional structures, and it repairs loop control and syntax issues that can cause evaluation failures.



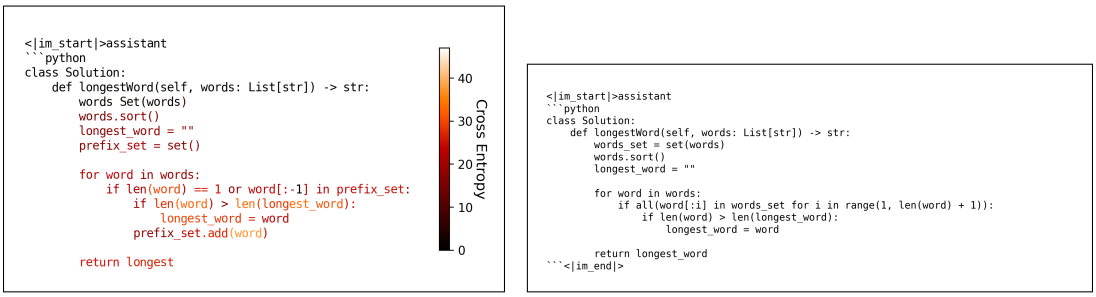
(a) Original response (b) Refined response

Figure 8: The reward of the original response is -0.3, while that of the refined response is 1.0. The original response also misses the initial signal `python`, which is correctly restored in the refined version.



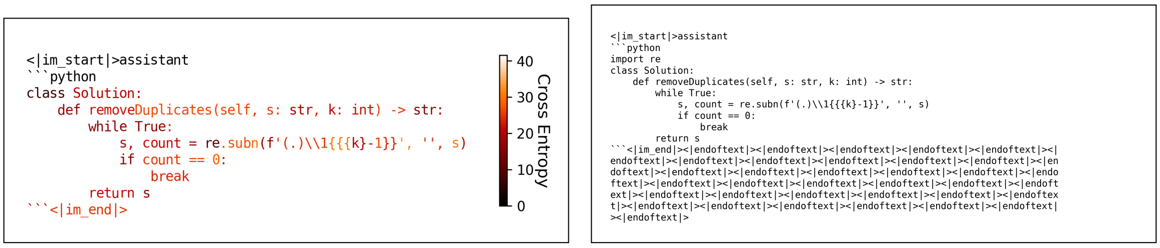
(a) Original response (b) Refined response

Figure 9: The reward of the original response is -1.0, while that of the refined response is 1.0. The refined response eliminates redundant operations and enforces a clearer logical structure.



(a) Original response (b) Refined response

Figure 10: The reward of the original response is -1.0, while that of the refined response is 1.0. The original response produces an incorrect computation branch, which the refined response corrects by revising the conditional structure.



(a) Original response

(b) Refined response

Figure 11: The reward of the original response is -0.6, while that of the refined response is 1.0. The refined version improves loop control and resolves syntax inconsistencies present in the original code.

D RFT from the Mirror Learning Perspective

Definition 1. *A drift functional $\mathfrak{D}$ is a map*

$$\mathfrak{D} : \Pi \times \mathcal{S} \to \{\mathfrak{D}_\pi(\cdot|s) : \mathcal{P}(\mathcal{A}) \to \mathbb{R}\},$$

such that for all $s \in \mathcal{S}$, and $\pi, \bar{\pi} \in \Pi$, writing $\mathfrak{D}_\pi(\bar{\pi}|s)$ for $\mathfrak{D}_\pi(\bar{\pi}(\cdot|s)|s)$, the following conditions are met

1. *$\mathfrak{D}_\pi(\bar{\pi}|s) \geq \mathfrak{D}_\pi(\pi|s) = 0$ (nonnegativity),*
2. *$\mathfrak{D}_\pi(\bar{\pi}|s)$ has zero gradient[6] with respect to $\bar{\pi}(\cdot|s)$, evaluated at $\bar{\pi}(\cdot|s) = \pi(\cdot|s)$ (zero gradient).*

Definition 2. *We say that $\mathcal{N} : \Pi \to \mathbb{P}(\Pi)$ is a neighbourhood operator, where $\mathbb{P}(\Pi)$ is the power set of Π, if*

1. *It is a continuous map (continuity),*
2. *Every $\mathcal{N}(\pi)$ is a compact set (compactness),*
3. *There exists a metric $\chi : \Pi \times \Pi \to \mathbb{R}$, such that $\forall \pi \in \Pi$, there exists $\zeta > 0$, such that $\chi(\pi, \bar{\pi}) \leq \zeta$ implies $\bar{\pi} \in \mathcal{N}(\pi)$ (closed ball).*

The trivial neighbourhood operator is $\mathcal{N} \equiv \Pi$.

The core theoretical insight of mirror learning is as follows: if the conditions specified in the definition are satisfied, and a better policy lies within the defined neighborhood $\mathcal{N}(\pi_{\text{old}})$, then applying the mirror learning update (as shown in Equation 1) guarantees that the value function improves monotonically for all states s, i.e., $V\pi_{\text{new}}(s) \geq V\pi_{\text{old}}(s), \forall s \in S$. Since $V(s)$ is bounded, this monotonic improvement ensures that the policy converges to the optimal one.

Under the mirror learning framework, a static constraint such as the KL regularization can be interpreted as defining a fixed neighborhood $\mathcal{N}(\pi_0)$. When the policy update is small, this constraint may have little effect. However, as the update magnitude increases, it may become increasingly difficult to find an improved policy within $\mathcal{N}(\pi_0)$, thereby hindering further optimization.

In contrast, the dynamic constraint we propose replaces this static neighborhood with $\mathcal{N}(\pi_{\text{refiner}})$, which helps mitigate this limitation. Assuming that π_{refiner} can effectively track the evolution of π_θ, it becomes more likely that improved policies remain accessible within the neighborhood. This property supports continuous policy improvement and facilitates convergence to the optimal policy.

E Limitations

The efficacy of the dynamic constraint paradigm is intrinsically coupled with the refinement capabilities of the reference model. While our filtering mechanism ensures training stability by discarding failed corrections, it necessarily introduces a trade-off in constraint density and robustness. Furthermore, for complex reasoning tasks that demand extremely long output trajectories, the autoregressive nature of the refiner presents a non-trivial challenge in maintaining global structural coherence. We anticipate that these limitations will naturally recede as the frontier of model capacity continues to advance. In the future, we plan to extend this framework by introducing a block-wise refinement mechanism, which we believe will be essential for scaling dynamic constraints to the most demanding reasoning and agentic tasks.

[6]More precisely, all its Gâteaux derivatives are zero.