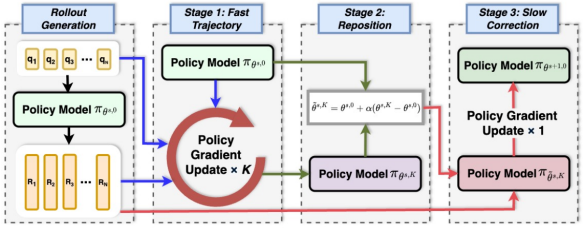
Slow-Fast Policy Optimization: Reposition-Before-Update for LLM Reasoning

Ziyan Wang*[1], Zheng Wang*[2], Xingwei Qu, Qi Cheng, Shengpu Tang, Minjia Zhang[2], Xiaoming Huo[1]

[1]Georgia Institute of Technology, [2]SSAIL Lab, University of Illinois Urbana-Champaign, *Equal contribution



Background

- Methods such as GRPO rely on **single-batch rollout** sampling to estimate policy gradients, making their effectiveness highly dependent on the quality of sampled trajectories.
- In practice, rollout generation is both **computationally expensive and statistically inefficient**, as each trajectory is used only once under strict on-policy constraints.
- The stochastic nature of rollouts introduces **high variance in gradient estimation**, particularly in early training when policies are still poor calibrated.

Motivation

- Improving training efficiency requires going beyond single-batch ono-policy updates, but doing so risks introducing distribution mismatch.
- Existing approaches either strictly follow on-policy updates with limited learning signal, or reuse trajectories at the cost of stability.
- This reveals a fundamental tension between on-policy consistency and efficient utilization of trajectory information.

Slow-fast Policy Optimization

Algorithm 1 SFPO: unified fast–reposition–slow update.

Require: Initial policy $\pi_{\theta^{0,0}}$, dataset $\mathcal{D}$, hyperparameters $S, K, \alpha_0, \eta, \omega, \tau$, loss $\mathcal{L}(\theta)$
Initialize rolling buffer $\mathcal{H} \leftarrow \emptyset$, stats $(\mu, \sigma) \leftarrow (0, 1)$, trigger index $s^\star \leftarrow +\infty$
for $s = 0, 1, \ldots, S-1$ **do**
  Generate rollouts with the current policy $\pi_{\theta^{s,0}}$ on prompts from $\mathcal{D}$.
  $\alpha \leftarrow \alpha_0 \cdot \mathbf{1}[\, s < s^\star \,]$ *// Set α for this iteration from past trigger (non-anticipatory)*
  if $\alpha = 0$ **then**
    $\theta^{s,K} \leftarrow \theta^{s,0}$ *// Skip fast trajectory & reposition*
  else
    for $k = 0, 1, \ldots, K-1$ **do**
      $\theta^{s,k+1} \leftarrow \theta^{s,k} - \eta \nabla_\theta \mathcal{L}(\theta^{s,k})$ *// Stage I: Fast Trajectory*
    end for
    $\tilde{\theta}^{s,K} \leftarrow \theta^{s,0} + \alpha(\theta^{s,K} - \theta^{s,0})$ *// Stage II: Reposition*
  end if
  $\theta^{s+1,0} \leftarrow \tilde{\theta}^{s,K} - \eta \nabla_\theta \mathcal{L}(\tilde{\theta}^{s,K})$ *// Stage III: Slow Correction*
  Compute entropy H_s; update rolling buffer $\mathcal{H}$ (keep last ω ones) and (μ_s, σ_s).
  $Z_s \leftarrow \frac{H_s - \mu_s}{\sigma_s + \varepsilon}$ *(ε for numerical stability)*
  if $s^\star = +\infty$ **and** $|Z_s| \geq \tau$ **then**
    $s^\star \leftarrow s + 1$ *// will set $\alpha = 0$ for all future $s' \geq s^\star$*
  end if
end for
return final policy $\pi_{\theta^{S,0}}$

- **Stage I: Fast Trajectory**
  In standard on-policy policy-gradient methods, each step is updated by a single stochastic gradient, suffering from high variance. SFPO mitigates this by performing a short fast trajectory of k inner updates:

$$\theta^{s,k+1} = \theta^{s,k} - \eta \nabla_\theta \mathcal{L}(\theta^{s,k}), \qquad k = 0, \ldots, K-1.$$

- **Stage II: Reposition**
  While the fast trajectory of Stage I improves stability, it also changes the nature of the update from on-policy to off-policy. SFPO introduces a reposition step that interpolates that fast trajectory back toward its stating point:

$$\tilde{\theta}^{s,K} = \theta^{s,0} + \alpha(\theta^{s,K} - \theta^{s,0}), \qquad \alpha \in [0, 1].$$

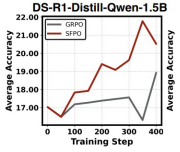
- **Stage III: Slow Correction.**
  SFPO further applies a slow correction aligned with the local curvature at the update point:

$$\theta^{s+1} = \tilde{\theta}^{s,K} - \eta \nabla_\theta \mathcal{L}(\tilde{\theta}^{s,K}).$$

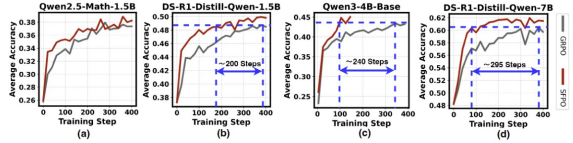
Performance on Math and Code Tasks

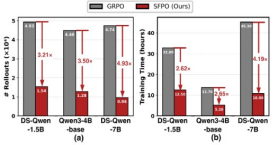
Model	Method	Math-500	AIME24	AIME25	AMC	Minerva	Olympiad	Avg
Qwen2.5-Math-1.5B	Base	55.55	9.17	5.83	37.65	17.74	28.45	25.73
	GRPO	77.15	16.67	11.67	53.31	31.89	39.42	38.35
	SFPO	**78.35**	**20.00**	**15.00**	**56.02**	**32.07**	**39.72**	**40.19**
Qwen2.5-Math-7B	Base	71.65	21.67	9.17	53.61	27.02	38.54	36.94
	GRPO	82.50	34.20	20.83	71.08	36.76	44.80	48.36
	SFPO	**82.30**	**35.00**	**20.83**	**74.49**	**36.94**	**45.59**	**49.19**
DS-distilled-Qwen-1.5B	Base	73.80	16.67	20.00	47.59	28.86	36.94	37.31
	GRPO	84.65	30.00	23.33	66.86	31.71	49.85	47.73
	SFPO	**86.10**	**32.50**	**30.83**	**70.28**	**32.81**	**50.67**	**50.53**
DS-distilled-Qwen-7B	Base	83.60	28.33	25.00	61.75	41.73	48.89	48.21
	GRPO	91.7	50.00	35.83	80.42	43.65	61.24	60.47
	SFPO	**92.60**	**54.17**	**37.50**	**83.75**	**44.49**	**65.73**	**63.04**
Qwen3-4B-Base	Base	45.25	2.50	0.83	20.48	15.99	20.66	17.62
	GRPO	83.35	16.67	17.50	**59.03**	38.78	48.59	43.99
	SFPO	**84.30**	**21.67**	**20.83**	57.23	**40.81**	**48.67**	**45.59**

- **Consistent improvements** across models and scales (1.5B to 7B).
- **Strong gains** on challenging reasoning tasks (e.g., up to +7.5 on AIME 25).
- **Generalizes beyond math to code tasks:** SFPO improves code generation performance (e.g., 18.91 -> 21.77), demonstrating broader applicability.



More Efficient Training





- **4–5×** reduction in rollout usage.
- **Up to 4.19×** end-to-end speedup with no extra memory cost

Corresponding authors: wzy@gatech.edu, zhengw10@illinois.edu

Project Website: https://slow-fast-po.github.io/