Efficiently Learning To Reason or Not to Reason: Root-token Policy Optimization for Adaptive Thinking

Taehyeon Kim[♠] **Hyunsoo Lee**[♠] **Youngsoo Jang**[‡,†,*] **Moontae Lee**[♠,§,†]

[♠]LG AI Research [‡]UNIST [§]University of Illinois Chicago

Abstract

Large reasoning models (LRMs) achieve strong performance by externalizing explicit reasoning traces before producing the answer, yet suffer from *overthinking* challenge that allocates uniformly heavy computation to queries of varying difficulty. While proprietary models mitigate this via opaque routing, open-source LRMs still lack an efficient mechanism to internalize adaptive reasoning due to both expensive training cost and limited disclosure of training recipes. In response, we introduce **RPO** (Root-token Policy Optimization), a framework that enables LRMs to self-determine when to reason by training only the initial root token (e.g., whether to invoke the <think> tag) via group relative reward and group-wise advantages. By focusing on this pivotal branching point, **RPO** drastically reduces training overhead and VRAM usage. Across multiple model families and scales, **RPO** learns difficulty-aware adaptive thinking at just ∼2% of the training compute of prior adaptive-reasoning methods.

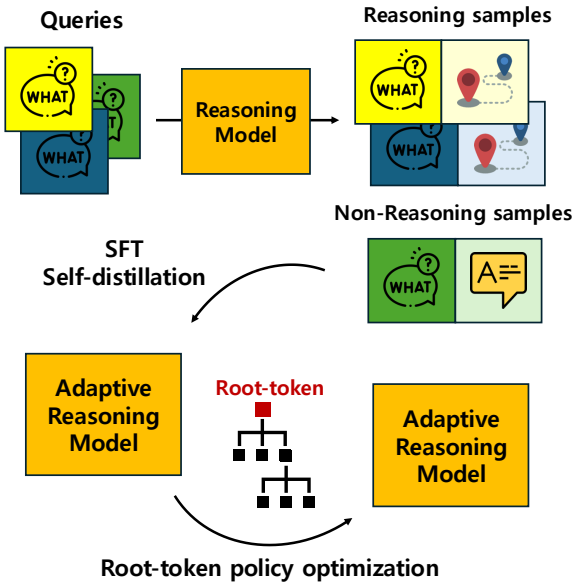


Figure 1: Overview of our training pipeline. We generate paired reasoning samples and non-reasoning samples from input queries through the original reasoning model. These samples are self-distilled to manifest adaptive reasoning. Finally, root-token policy optimization (**RPO**) is applied to the model, refining the policy by updating only the initial branching token (i.e., root-token) to improve difficulty-aware adaptive reasoning.

1 Introduction

The advent of large reasoning models (LRMs) has redefined the benchmark requiring complex reasoning (Guo et al., 2025; Jaech et al., 2024). The efficacy of LRMs has been attributed to the additional computation allocated at test time through explicit reasoning paths, typically encapsulated within delimiters such as <think> ... </think>. By exploring logical traces, reflecting on internal errors, and self-verifying intermediate steps, these models have achieved breakthroughs in mathematical and symbolic reasoning tasks. These days, this *think-before-answer* paradigm has emerged as the de facto standard.

Despite the potential of LRMs, there remains a significant limitation that general LRMs tend to apply this exhaustive explicit reasoning uniformly across all queries, regardless of their intrinsic complexity. This results in the *overthinking* problem (Feng et al., 2025; Sui et al., 2025) where substantial computational resources and latency are wasted on trivial questions. Consequently, a surge of emerging research has been driven into striking a balance between reasoning quality and computational efficiency (Zhang et al., 2025; Fang et al., 2025; Wu et al., 2025; Wang et al., 2025b). While proprietary services such as GPT5 (OpenAI, 2025) and Claude (Anthropic, 2024) rely on black-box routing to mitigate overthinking, a novel model-centric alternative is required to internalize adap-

[*]Work completed while employed at LG AI Research.
[†]Corresponding authors.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 17934–17949

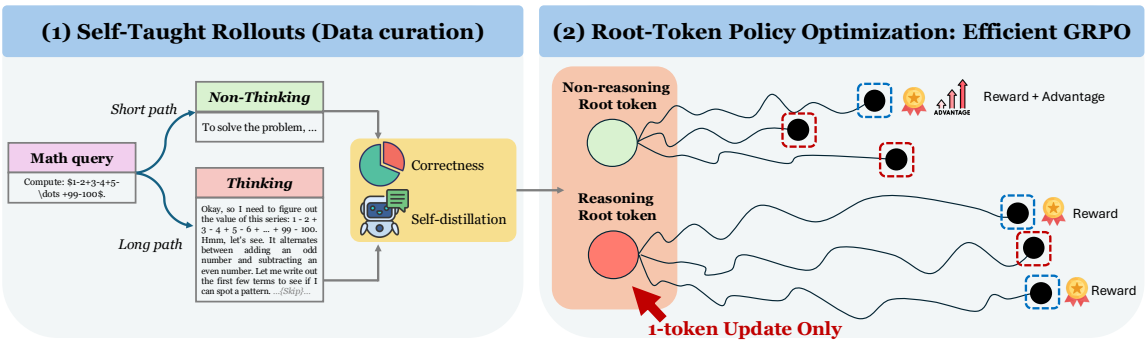


Figure 2: Detailed overview of our training framework. After supervised finetuning (SFT) with self-generated correct samples, **RPO** involves updates only the initial decision token with group normalized rewards and advantages.

tive reasoning via policy optimization with open dataset for the open-source community.

While adaptive reasoning models that learn when to reason have been explored via policy optimization, these studies have faced the following challenges:

1. Training inefficiency. Optimizing over long context trajectories with policy optimization incurs prohibitive VRAM and temporal costs, hindering accessible and iterative development.
2. Data curation. There is a lack of standardized methods for *baking* difficulty-aware datasets that effectively balance reasoning traces with direct responses across heterogeneous domains.
3. Initialization gap. Prior works train from scratch or adapt base models, often avoiding adapting the original LRM due to its training sensitivity.

To alleviate these challenges, we propose **RPO** (Root-token Policy Optimization), an efficient training method that enables adaptive reasoning by learning only the initial root token (e.g., the decision to open or close the `<think>` tag). Our method involves two stages (Figure 1): (1) offline synthesis which generates paired reasoning and non-reasoning responses via forced chat templates, and (2) **RPO**, which restricts gradient updates to the pivotal branching decision. Our design is complementary to recent token-selective RL that updates only a small subset of decision tokens (e.g., high-entropy forking tokens) rather than the full trajectory (Wang et al., 2025a); **RPO** pushes this idea to the extreme by optimizing only the root routing token (Figure 2). We demonstrate that **RPO** drastically enhances memory and compute efficiency during training while preserving the model's original capability.

Our contributions are as follows:

- We propose **RPO**, a novel framework, termed as root-token optimization to enable adaptive reasoning by optimizing only the initial root-token whether to reason, significantly reducing training overhead as well as efficient difficulty-aware routing (§3).
- Our empirical results demonstrate **RPO** enables LRMs learning when to reason over diverse LRM families and scales. We also provide an analysis on calibration for an explicit reasoning knob (e.g., target Think% or routing threshold for root-token probability) (§4, §5).
- We demonstrate that difficulty-aware policies in mathematical domains elicit adaptive reasoning in broader scientific and code domain (§5).

2 Related work

2.1 Efficient reasoning models

LRMs allocate additional test-time computation to explicit reasoning traces often with specific delimiters such as `<think>...</think>` (Guo et al., 2025; Jaech et al., 2024). This *think-before-answer* format enables iterative exploration and self-verification, but it is frequently applied *uniformly* across inputs. Such uniform long-form reasoning leads to the *overthinking* challenge with high latency and compute on intrinsically easy queries and sometimes producing redundant reasoning tokens (Feng et al., 2025; Sui et al., 2025).

A broad line of work has pursued efficiency via *adaptive computation* and *early stopping*. Classical dynamic-computation methods and early-exit approaches terminate generation once confidence criteria are met, reducing unnecessary computation

without changing the core model objective (Schuster et al., 2022; Bae et al., 2025). Recent reasoning-focused extensions similarly aim to avoid excessive reasoning tokens by learning or heuristically determining when further context is unlikely to help, and by truncating or suppressing low-information reasoning segments (Sui et al., 2025).

Another direction controls reasoning *length* through prompt- or budget-based strategies. Methods that rely on fixed or predicted token budgets can reduce verbose CoT while maintaining accuracy, but often require external budget selection rules or calibrated estimators, which may not transfer cleanly across domains (Han et al., 2024). In parallel, internal-rationale approaches such as Quiet-STaR (Zelikman et al., 2024) encourage models to form latent rationales during generation, while simplified recipes such as s1 (Muennighoff et al., 2025) show that carefully constructed 1k data and prompting can recover much of test-time scaling benefits with minimal training overhead.

As an orthogonal axis over these approaches, our goal is not to enforce a budget or modify the architecture, but to *internalize the gating decision itself* whether to open a long explicit reasoning trace in the first place. **RPO** optimizes only the *root token* that triggers the `<think>` branch, enabling adaptive reasoning with minimal training footprint and without backpropagating over whole reasoning trajectories.

2.2 Difficulty-aware policy optimization

The question of *learning when to reason* has recently been reframed as *difficulty-aware policy optimization*: the model should allocate explicit reasoning only when needed, while responding directly to easy or non-reasoning queries. This direction is especially relevant under the explicit `<think>` styled format, where the core idea is a discrete *branching action* that enters a long reasoning mode versus produces a direct response (Zhang et al., 2025; Fang et al., 2025; Wu et al., 2025; Wang et al., 2025b).

Representative methods learn such policies through RL or preference learning. AdaptThink (Zhang et al., 2025) trains a policy to select *thinking* versus *non-thinking* modes conditioned on difficulty, while Thinkless (Fang et al., 2025) uses preference/RL signals to reduce redundant thinking tokens in general conversations. Beyond binary gating, ARM (Wu et al., 2025) expands the action space to multiple reasoning formats (e.g., direct answer, short CoT, long CoT, code), and AdaReasoner (Wang et al., 2025b) treats reasoning configurations as actions in a model-agnostic RL framework. Together, these works show that adaptive reasoning is learnable and can outperform static long-CoT inference.

However, existing policy optimization approaches face practical hurdles that motivate our design. First, many studies are benchmark-centric and largely evaluated on math-heavy suites, leaving open how well their learned routing *calibrates* (e.g., whether the thinking rate remains aligned with difficulty on open-domain and scientific queries). Second, optimizing policies over whole long reasoning trajectories is often *training-cost dominated*: GRPO-style post-training incurs expensive cost of VRAM, gradient computation, and wall-clock costs (Guo et al., 2025), also detailed in §3.3 and Table 5. Third, effective training frequently depends on curated external prompts, data from larger teacher models, paired supervision, or proprietary query streams for routing signals, and these factors are rarely opened to the community.

3 Method

3.1 Motivation

Adaptive reasoning is fundamentally a *routing* problem. Before generating an answer, the model must decide whether to spend tokens on explicit deliberation. In open-domain tasks, supervising this routing is difficult because task *difficulty* is latent: correctness can be ambiguous and the same query may be easy or hard depending on a model's calibration and knowledge. As a result, fixed heuristics (e.g., always long CoT or static budgets) are often miscalibrated.

Why math as a proxy We ground our training signal in mathematical reasoning because it admits *deterministic* verification and a *model-relative* notion of when reasoning pays off. For each input query x, we elicit two mode-conditioned behaviors: (i) a direct-answer mode (SHORT) and (ii) an explicit-reasoning mode (LONG). Let $c_S(x), c_L(x) \in \{0, 1\}$ denote exact correctness under each mode. Rather than relying on external difficulty labels, we can use the difference between mode-conditioned success rates (e.g., Pass@k) as an empirical difficulty proxy: instances where LONG is more reliable than SHORT are treated as harder for the current model, and vice versa. This model-specific signal provides supervision for

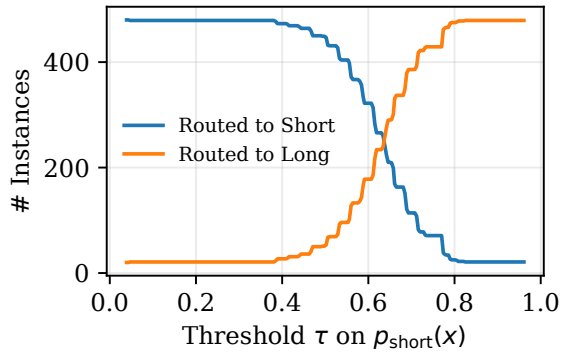


Figure 3: Thresholding on a SHORT-mode confidence signal yields a smooth routing trade-off about probability threshold vs number of routed instances over MATH500 dataset (Hendrycks et al., 2021).

learning routing policies without external difficulty labels.

Self-taught difficulty signals After a lightweight SFT step that teaches the model to follow our SHORT/LONG templates (§3.2), we observe that routing can be anticipated at the generation root. We obtain a model-specific difficulty signal by contrasting mode-conditioned accuracy on the same query (e.g., Pass@k under SHORT vs. LONG). To expose this signal for routing, we *parameterize* the mode decision at the generation root: under our template, the first newline token(s) after `<think>` select LONG vs. SHORT (`\n` vs. `\n\n`). Let z_S denote the SHORT root token (`\n\n`) and define a root-level SHORT-preference proxy as the model's conditional probability at this decision point:

$$p_{\text{short}}(x) \;=\; p_\theta(z_S \mid x, \texttt{<think>})\,.$$

We route by thresholding:

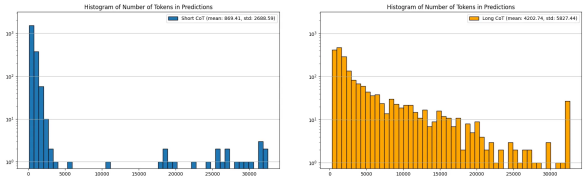
$$\pi_\tau(x) = \begin{cases} \textsc{Short} & \text{if } p_{\text{short}}(x) \geq \tau, \\ \textsc{Long} & \text{otherwise.} \end{cases}$$

Varying τ yields a simple calibration knob (Figure 3): larger τ makes the router more conservative, assigning fewer queries to SHORT and increasing the fraction routed to LONG. This synthetically produces an interpretable accuracy–routing trade-off, where intermediate thresholds preserve overall accuracy while reducing LONG usage (Table 1).

Beyond accuracy, routing is motivated by *compute*. We measure generation length $L(x)$ (output tokens) and find that LONG exhibits a heavy-tailed length distribution (Figure 4), i.e., a small fraction of queries trigger disproportionately long traces. This makes the routing decision directly translate into cost control, and motivates learning a lightweight gating policy at the root that reduces unnecessary LONG usage without the loss of the model capability.

Method (τ)	SHORT	LONG	Overall
Original	0.00 (0)	83.6 (500)	83.6 (500)
Adaptive (0.75)	93.6 (62)	82.4 (438)	83.8 (500)
Adaptive (0.70)	89.7 (155)	78.8 (345)	82.2 (500)

Table 1: Accuracy (%) and routed instance counts (in parentheses) under threshold routing over MATH500 dataset (Hendrycks et al., 2021) with R1-1.5B model.



(a) SHORT (b) LONG

Figure 4: Token-length histograms (log-scaled frequency) for SHORT and LONG mode.

3.2 SFT for mode selection

Mode definition We define two generation modes using only the reasoning-tag template at the generation root. The token `<think>` is treated as a special token, and `\n` vs. `\n\n` are tokenized into different root tokens. Concretely, we use:

- LONG: prefix `<think>\n` so that the model continues with explicit reasoning content.
- SHORT: prefix `<think>\n\n</think>\n\n` to suppress explicit reasoning.

Self-generated trajectories from OpenR1-Math queries We construct SFT supervision from OpenR1-Math queries by sampling multiple generations from the given LRM under each mode template (SHORT/LONG). We retain only generations that (i) satisfy the mode-specific template and (ii) match the exact answer, yielding two pools of correct demonstrations, $\mathcal{S}_S$ and $\mathcal{S}_L$. We then randomly sample K instances from each pool for SFT. We do not enforce that a problem must be correct in both modes.

SFT We use a SFT loop (1) to manifest mode-conditioned *style* without performance degradation under the default chat template. Let $\text{Acc}_{\text{orig}} = \text{Eval}(\pi_0; \textsc{DefaultChat})$ be the initial accuracy, and let $\text{Acc}(\pi) = \text{Eval}(\pi; \textsc{DefaultChat})$

Algorithm 1 SFT for mode selection

Require: base model π_0, prompts $\mathcal{D}$, samples per prompt M, tolerance ϵ, budget K

$a_0 \leftarrow \mathrm{Eval}(\pi_0;\ \textsc{DefaultChat})$ ▷ default template
$\pi \leftarrow \pi_0$
while $\left|\mathrm{Eval}(\pi;\ \textsc{DefaultChat}) - a_0\right| \le \epsilon$ **do**
  $\mathcal{S}_S \leftarrow \mathrm{CollectCorrect}(\pi, \mathcal{D}, M;\ \textsc{ShortPrefix})$
  $\mathcal{S}_L \leftarrow \mathrm{CollectCorrect}(\pi, \mathcal{D}, M;\ \textsc{LongPrefix})$
  $\mathcal{T} \leftarrow \mathrm{Sample}(\mathcal{S}_S, K) \cup \mathrm{Sample}(\mathcal{S}_L, K)$
  $\pi \leftarrow \mathrm{SFT}(\pi, \mathcal{T})$
end while

denote the same metric for an SFT candidate π. We iteratively refresh the correctness-filtered SFT dataset and continue training only while the performance gap remains within a tolerance ϵ:

$$\left|\mathrm{Acc}(\pi) - \mathrm{Acc}_{\mathrm{orig}}\right| \le \epsilon.$$

The loop is intentionally general. Empirically, across LRMs we tested, a single iteration typically sufficed (i.e., the constraint was satisfied without repeated resampling).

3.3 Root-token policy optimization (RPO)

SFT establishes SHORT/LONG behaviors, but reliable *adaptive selection* requires directly optimizing the gating policy. The post-SFT model already knows *how* to produce either style; what remains is learning *when* to invoke each one. We therefore view adaptive reasoning as a *branching* problem: once the gate is chosen, the continuation is simply a mode-conditioned rollout rather than a learning target. Motivated by this hypothesis, we apply **RPO** only to the first gating token (the root), treating the remainder of each sampled trajectory purely as environment output.

Root-token action space We fix a <think> prefix and parameterize gating by the first newline tokenization:

$$z \in \{\texttt{\textbackslash n},\ \texttt{\textbackslash n\textbackslash n}\}.$$

Because <think> is a special token and \n vs. \n\n are distinct tokens, z is a well-defined categorical action. Selecting $z = \texttt{\textbackslash n}$ enters LONG: the model continues generating inside the <think> block (explicit reasoning). Choosing $z = \texttt{\textbackslash n\textbackslash n}$ enters SHORT: the template immediately closes the thinking block, i.e., <think>\n\n</think>\n\n. **RPO** optimizes only $p_\theta(z \mid x)$; subsequent tokens are masked from the gradient.

Balanced rollouts For each query x, we collect n rollouts with a fixed split: $n/2$ SHORT and $n/2$ LONG. Let $C_S(x)$ and $C_L(x)$ be the numbers of correct rollouts within the SHORT and LONG groups, respectively. This balanced sampling prevents trivial mode collapse and yields comparable advantage statistics across modes.

Algorithm 2 Root-token policy optimization

Require: policy π_θ (init from SFT), reference π_{ref} (frozen), prompts $\mathcal{D}_{\text{rl}}$, rollouts per prompt n (even), context budget S, temperature τ, KL weight β, mode weights (α_S, α_L)

Define gating tokens: $z \in \{\texttt{\textbackslash n},\ \texttt{\textbackslash n\textbackslash n}\}$
Define templates: SHORT(z=\n\n), LONG(z=\n)
for each training step **do**
  Sample a minibatch $\mathcal{B} \subset \mathcal{D}_{\text{rl}}$
  for each prompt $x \in \mathcal{B}$ **do**
    Balanced rollouts: generate $n/2$ trajectories with SHORT template and $n/2$ with LONG
    $\{(z_i^{(x)}, y_i^{(x)})\}_{i=1}^{n} \leftarrow \mathrm{Rollout}(\pi_\theta, x;\ S, \tau)$ ▷ $z_i^{(x)}$ is the root newline token(s)
    Compute correctness indicators $c_i^{(x)} \leftarrow \mathbf{1}\{\mathrm{Ans}(y_i^{(x)}) \text{ is correct}\}$
    Let $C_S(x) \leftarrow \sum_{i:\, z_i^{(x)} = \texttt{\textbackslash n\textbackslash n}} c_i^{(x)}$, $C_L(x) \leftarrow \sum_{i:\, z_i^{(x)} = \texttt{\textbackslash n}} c_i^{(x)}$
    for $i = 1$ to n **do**
      $r_i^{(x)} \leftarrow \alpha_S \cdot C_S(x)$ if $z_i^{(x)} = \texttt{\textbackslash n\textbackslash n}$ else $r_i^{(x)} \leftarrow \alpha_L \cdot C_L(x)$
    end for
    Store root-token log-probs $\ell_i^{(x)} \leftarrow \log \pi_\theta(z_i^{(x)} \mid x)$ and $\ell_i^{(x),\text{ref}} \leftarrow \log \pi_{\text{ref}}(z_i^{(x)} \mid x)$
  end for
  Per-mode group-relative advantages :
  $\mathcal{I}_m \leftarrow \{(x,i) : x \in \mathcal{B},\ m_i^{(x)} = m\}$
  $\mu_m \leftarrow \frac{1}{|\mathcal{I}_m|}\sum_{(x,i)\in\mathcal{I}_m} r_i^{(x)}$, $\sigma_m^2 \leftarrow \frac{1}{|\mathcal{I}_m|}\sum_{(x,i)\in\mathcal{I}_m} (r_i^{(x)} - \mu_m)^2$
  $A_i^{(x)} \leftarrow (r_i^{(x)} - \mu_{m_i^{(x)}})/(\sigma_{m_i^{(x)}} + \varepsilon) \quad \forall (x,i)$
  Root-token-only update (mask all tokens except z):
  $\theta \leftarrow \mathrm{Update}\Big(\theta,\ \nabla_\theta\big[-\frac{1}{|\mathcal{B}|}\sum_{x\in\mathcal{B}} \frac{1}{n}\sum_{i=1}^{n} A_i^{(x)}\, \ell_i^{(x)} + \beta \cdot \mathrm{KL}(\pi_\theta(z \mid x) \,\|\, \pi_{\text{ref}}(z \mid x))\big]\Big)$
end for

Prefix-level reward For a fixed prompt x, we refer to the $n/2$ rollouts that share a forced mode as the *per-prompt mode group* (one for SHORT, one for LONG). Each rollout is rewarded by the success count of its per-prompt mode group:

$$r_i^{(x)} = \begin{cases} \alpha_S \cdot C_S(x) & \text{if rollout } i \text{ is in SHORT,} \\ \alpha_L \cdot C_L(x) & \text{if rollout } i \text{ is in LONG.} \end{cases}$$

We set α_L=1 and use a mild SHORT preference α_S=$1 + \frac{1}{n}$ to reflect the objective of saving compute without sacrificing the correctness. Because $r_i^{(x)}$ depends on the per-prompt mode-group outcome, the update favors routing toward the mode that yields higher group-level correctness for that prompt.

Model	Method	Updated tokens T	KV / rollout (GiB)	Act(bp) / rollout (GiB)	Train+Opt (GiB)	Peak @ B=8 (GiB)
1.5B	Original GRPO	12K	2.38	13.18	22.35	146.86 (1.00×)
	RPO	1	2.38	1.10×10^{-3}	22.35	41.40 (0.28×)
	RPO with LoRA update	1	2.38	1.10×10^{-3}	0.07	19.12 (0.13×)
7B	Original GRPO	12K	6.35	35.16	104.31	436.34 (1.00×)
	RPO	1	6.35	2.93×10^{-3}	104.31	155.11 (0.36×)
	RPO with LoRA update	1	6.35	2.93×10^{-3}	0.21	51.01 (0.17×)

Table 2: Peak VRAM (GiB) for long-context GRPO with uniform context S=13K (1K prompt+12K generation) and B=8 concurrent rollouts. **RPO** gains memory efficiency by updating the root token (T=1). LoRA further reduces trainable-state memory with fewer trainable parameters.

Per-mode group-relative advantages To optimize *switching* under the enforced split, we standardize advantages within each mode by pooling rewards. Per-prompt-mode would be degenerate: by construction, $r_i^{(x)} = \alpha_{m_i} C_{m_i}(x)$ is constant across the $n/2$ rollouts that share a mode within prompt x, so the within-mode-group variance is identically zero. Cross-prompt variation in $C_S(x)$ and $C_L(x)$ is what supplies the non-degenerate variance needed for advantage estimation. Concretely, for rollout i of prompt x in mode $m_i^{(x)} \in \{S, L\}$, let $\mathcal{I}_m = \{(x, i) : x \in \mathcal{B},\ m_i^{(x)} = m\}$ denote the *group-level mode pool*. Then,

$$A_i^{(x)} = \frac{r_i^{(x)} - \mu_{m_i^{(x)}}}{\sigma_{m_i^{(x)}} + \varepsilon},$$
$$\mu_m = \frac{1}{|\mathcal{I}_m|} \sum_{(x,j)\in\mathcal{I}_m} r_j^{(x)}, \quad \sigma_m^2 = \frac{1}{|\mathcal{I}_m|} \sum_{(x,j)\in\mathcal{I}_m} (r_j^{(x)} - \mu_m)^2.$$

Pooling is *per-mode* (separate μ_S, σ_S and μ_L, σ_L) so that the standardized advantages remain on a comparable scale even when the two modes have systematically different success rates; pooling is *group-level* so that within-mode variance is nonzero. The signal each $A_i^{(x)}$ encodes is therefore "how does prompt x's mode-m success compare to the group-wide mode-m baseline," which the policy converts into an updated routing distribution over z. This per-mode group relative scheme presupposes prompt-difficulty diversity within $\mathcal{B}$.

RPO objective with KL regularization We regularize the log-probability of the root action z with KL-to-reference loss:

$$\max_\theta \ \mathbb{E}_x \left[\frac{1}{n} \sum_{i=1}^{n} A_i \ \log p_\theta(z_i \mid x) \right] - \beta \ \mathrm{KL}(p_\theta(z \mid x) \,\|\, p_{\theta_{\text{ref}}}(z \mid x)) \,. \quad (1)$$

where θ_{ref} is the post-SFT checkpoint. In implementation, we generate full trajectories to evaluate correctness and compute rewards, but retain only the root log-probabilities. Other non-root tokens are discarded from the training graph, and the update is performed solely on $p_\theta(z|x)$.

Balanced difficulty sampling We perform **RPO** on a compact set of ∼1K queries comprised of a part of s1k (Muennighoff et al., 2025) and compression dataset (Arora and Zanette, 2025). With the post-SFT model, We estimate mode-conditioned success rates and exclude items where LONG is rarely correct ($\mathrm{acc}_L < \tau_{\min}$=0.25). We then stratify the remaining queries into *easy* and *hard* regimes and sample them in a 1:1 ratio to stabilize updates and avoid mode collapse. Concretely, *easy* includes queries where SHORT is already competitive (e.g., $\mathrm{acc}_S \geq \tau_{\text{easy}}$=0.75), while *hard* includes queries where LONG is consistently preferable (e.g., $\mathrm{acc}_S < \mathrm{acc}_L \leq \tau_{\text{hard}}$ with τ_{hard}=0.5).

Why root-only updates are memory-efficient Both vanilla GRPO and **RPO** must work with KV-cache over the full rollout context length S during sampling. The training-time gap arises from backpropagation. Vanilla GRPO updates all generated tokens, whereas **RPO** masks the trajectory and updates only the root gating token (T=1). As a result, the dominant activation term scales with the number of updated tokens,

$$\mathcal{M}_{\text{bp}} \propto B \cdot T \cdot L \cdot d \cdot b \cdot \kappa,$$

where B is the number of rollouts processed concurrently, L and d are model depth and width, b is the activation precision (e.g., bf16), and κ aggregates saved intermediates (e.g., attention/MLP). Thus, setting T=1 removes the linear dependence on long CoT length. Table 2 quantifies this effect under long-context rollout scenarios, where activation memory dominates the feasibility boundary.

Training configuration Unless noted otherwise, we use three H100 GPUs in a node. Each iteration

Model	Stage	MATH500			AIME24			AIME25			Avg			
		P@1	#Tok	Think%	P@1	#Tok	Think%	P@1	#Tok	Think%	P@1	#Tok	Think%	RAR↓
Qwen3-1.7B	Baseline (LONG)	91.1	4,539	100.0	45.6	15,569	100.0	38.1	15,830	100.0	58.3	11,979	100.0	1.72
	Baseline (SHORT)	73.0	978	0.0	12.3	3,898	0.0	11.5	2,180	0.0	32.3	2,352	0.0	0.00
	+ SFT	85.5	3,767	69.8	34.4	12,558	72.9	30.4	12,648	75.6	50.1	9,658	72.8	1.45
	+ **RPO**	79.5	2,754	31.8	43.8	13,315	93.8	31.7	14,231	96.3	51.7	10,100	74.0	1.43
Qwen3-8B	Baseline (LONG)	96.6	4,777	100.0	74.4	14,184	100.0	64.8	16,950	100.0	78.6	11,970	100.0	1.27
	Baseline (SHORT)	84.1	1,118	0.0	25.4	5,646	0.0	20.8	3,159	0.0	43.4	3,308	0.0	0.00
	+ SFT	90.6	3,390	53.5	54.0	12,350	59.2	46.7	12,661	59.2	63.8	9,467	57.3	0.90
	+ **RPO**	90.5	2,794	25.0	52.9	12,784	56.4	42.5	12,029	53.1	62.0	9,202	44.8	0.72
Exaone4-1.2B	Baseline (LONG)	91.0	5,416	100.0	54.0	18,529	100.0	43.8	19,124	100.0	62.9	14,356	100.0	1.59
	Baseline (SHORT)	75.6	854	0.0	21.9	3,075	0.0	15.2	2,718	0.0	37.6	2,216	0.0	0.00
	+ SFT	84.6	3,015	40.9	37.3	9,524	39.8	31.3	10,562	49.4	51.1	7,700	43.4	0.85
	+ **RPO**	83.5	2,455	25.5	33.8	6,592	21.9	27.3	6,665	23.3	48.2	5,237	23.6	0.49

Table 3: Baseline → SFT mode selection → **RPO** on math benchmarks. Pass@1 (P@1) is the average accuracy (%) over 16 runs. #Tok is average generated tokens. Think% is the fraction routed to LONG. **Avg** columns are simple averages over (MATH500, AIME24, AIME25). **RAR** (↓) is the reasoning-per-accuracy ratio, defined as $\mathrm{RAR}_{\text{avg}} = \frac{t_{\text{avg}}}{a_{\text{avg}}+\varepsilon}$, where a_{avg} and t_{avg} are Avg Pass@1 and Avg Think% as fractions. Lower is better.

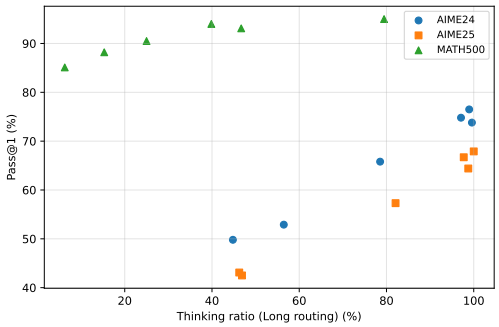


Figure 5: Think% versus Pass@1 on Qwen3-8B with **RPO**. Each point indicates an intermediate checkpoint.

samples 8 queries and collects n=16 rollouts per query (8 per mode) with temperature 0.6 and top-p 0.95. We allow maximum context length 16,384 for rollouts.

4 Experiment

4.1 Setup

Our pipeline follows **Baseline** → **SFT** → **RPO**, and we report (i) Pass@1 accuracy, (ii) average generated tokens, and (iii) the fraction routed to LONG (Think%) in Table 3. We report results on diverse models with different families and sizes: **Qwen3-1.7B**, **Qwen3-8B**, **Qwen3-32B** (Yang et al., 2025), **Exaone4-1.2B** (Research et al., 2025) and **R1-1.5B** (Guo et al., 2025). All models use a `<think>` style template and share the same two-mode gating mechanism implemented via the first newline token(s) after `<think>`.

4.2 Datasets and evaluation

Math benchmarks We evaluate on three standard math reasoning sets: MATH500 (Hendrycks et al., 2021), AIME24 (MAA, 2024), and AIME25 (MAA, 2025), scientific reasoning dataset as GPQA (Rein et al., 2024), and coding task with LiveCodeBench[1]. Following prior work, correctness is determined by exact match on the final answer parsed by `boxed{}`. At inference time, we match the original technical reports including temperature and sampling settings.

SFT data for mode selection We use mathematical queries from OpenR1-Math data (Open-R1, 2025) and self-generate candidate trajectories with the given model under each mode template. We filter candidates by correctness and basic template validity, producing two pools of correct demonstrations for SHORT and LONG. We then sample 20K instances per mode (40K total) for SFT, without requiring overlap between the two pools.

Implementation details We implement SFT and **RPO** in **OpenRLHF** (Hu et al., 2024), with a customized lightweight *asynchronous* rollout engine that overlaps generation and post-processing. Unless noted, we use max context **16K**, batch size **128**, and learning rate 10^{-6}. For **RPO**, we sample **16** rollouts per prompt (8/8 SHORT/LONG) and apply gradients only to the root gating token. We run $<$ **60** update steps for all model sizes.

4.3 Main results

Table 3 reports math performance across stages (**Baseline** → **SFT** → **RPO**). To summarize *reasoning efficiency* in one number, we use **RAR** (↓), the reasoning-per-accuracy ratio: $\mathrm{RAR}_{\text{avg}} = \frac{t_{\text{avg}}}{a_{\text{avg}}+\varepsilon}$,

[1]LiveCodeBench is sliced from March 25 2025 to April 15 2025

Method	MATH500		AIME24		Avg	
	P@1	Think%	P@1	Think%	P@1	RAR↓
Prior hybrid-reasoning baselines (reported in their tables)						
Original (LONG)	83.2	100.0	29.4	100.0	56.3	1.78
Original (SHORT)	67.2	0.0	14.0	0.0	40.6	0.00
Rejection Finetuning † (Zhang et al., 2025)	72.4	66.6	25.2	79.0	48.8	1.49
AdaptThink† (Zhang et al., 2025)	82.0	23.2	31.0	59.6	56.5	0.74
Thinkless† (Fang et al., 2025)	81.9	51.6	27.3	100.0	54.6	1.39
Ours (same backbone family; evaluated with our protocol)						
+ SFT	75.1	37.8	20.8	68.5	48.0	1.11
+ **RPO**	77.6	20.5	25.0	77.7	51.3	0.96

Table 4: Comparison with accuracy and Think% over R1-1.5B model. † presents the performance from the original paper.

where a_{avg} is Avg Pass@1 and t_{avg} is Avg Think% (both as fractions).

RPO improves the accuracy routing trade-off On the larger and mid-scale backbones, **RPO** reduces Think% with modest accuracy changes, yielding the best (lowest) RAR. For example, on Qwen3-8B, Avg Think% decreases from 57.3% to 44.8% while Avg Pass@1 stays close (63.8%→62.0%), improving RAR from 0.90 to 0.72; on Exaone4-1.2B, Think% drops from 43.4% to 23.6% and RAR improves from 0.85 to 0.49. On Qwen3-1.7B, the post-SFT operating point is already conservative on the LONG side (Think%=72.8, RAR=1.45); **RPO** yields a comparable operating point (Think%=74.0, RAR=1.43) that recovers accuracy via routing refinement rather than via further Think% compression.

SFT already enables routing; RPO refines it. SFT alone yields a strong reduction in Think% (and tokens), showing that mode-following is largely learned at the template level. **RPO** then fine-tunes the *root* decision to better match query difficulty. The direction of this refinement is configuration-dependent: on Qwen3-1.7B and R1-1.5B (Table 4), **RPO** recovers accuracy on top of SFT by reallocating LONG toward the harder subset; on Qwen3-8B and Exaone4-1.2B, **RPO** instead trades a small accuracy reduction for further Think% compression.

Comparison with other GRPO-based methods Table 4 compares **RPO** with recent hybrid reasoning baselines on the same R1 1.5B model. **RPO** lowers the fraction routed to LONG to 20.5% and reaches a two-task average accuracy of 51.3 with RAR=0.96. AdaptThink achieves a stronger absolute operating point (Avg P@1=56.5, RAR=0.74) but at a far larger training budget (Table 5); the comparison should therefore be read as two distinct points on a budget-accuracy frontier rather than a head-to-head ranking. Additionally, Table 5 highlights the compute-efficiency gap. Trajectory level GRPO is compute heavy because gradients traverse the full chain of thought. With matched accounting on an 8B reference model as an example, **RPO** reduces total training compute to about 0.02× AdaptThink.

5 Analysis

Controlling the degree of routing Figure 5 plots P@1 against Think% for intermediate **RPO** checkpoints. Points with comparable accuracy but different thinking ratios indicate multiple feasible operating points. Thus, calibration is practical: choose a checkpoint that satisfies a compute/routing budget while maintaining accuracy. The achievable frontier depends on the RL prompt set and easy/hard mix, so checkpoint selection is dataset-conditioned rather than universal.

LiveCodeBench and GPQA Table 6 extends evaluation beyond math to coding and scientific QA, where the optimal amount of thinking is more heterogeneous. Across models, SFT already introduces a meaningful reduction in explicit reasoning, while **RPO** further reshapes the accuracy routing profile by updating only the root decision. The effect is model-size dependent: on Qwen3-8B and Exaone4-1.2B, **RPO** compresses average decoding effort and lowers RAR; on Qwen3-32B, where SFT already produces a near-baseline operating point (Avg Think%=94.6, RAR=1.52), **RPO** yields a comparable but not strictly better RAR (1.55), reflecting that the room for further routing-only gains shrinks once the underlying solver is sufficiently strong and the SFT operating point is already conservative. Practically, this suggests treating the post-**RPO** checkpoint as a controllable router (with τ-sweeps) rather than as a fixed point that uniformly dominates SFT.

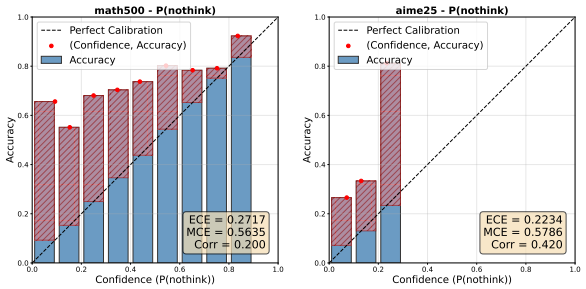
Calibration of root confidence Figure 6 reports reliability diagrams with Qwen3-1.7B model for the root SHORT confidence $p_{\text{short}}(x)=p_\theta(z_S|x,\texttt{<think>})$. We bin p_{short} into 10 uniform intervals and compute (i) **Corr**, the Pearson correlation between per query confidence and empirical accuracy, and (ii) calibration errors **ECE** $= \sum_b \frac{n_b}{N}|\text{acc}_b - \text{conf}_b|$ and **MCE** $= \max_b|\text{acc}_b - \text{conf}_b|$, where $\text{acc} * b$ and $\text{conf} * b$ are the average accuracy and confidence within bin b (Guo et al., 2017). Overall, $p_{\text{short}}(\cdot)$ tends

Method	SFT spec	GRPO spec	Forward PFLOPs	Backward PFLOPs	Total PFLOPs	Rel. Total (vs. AdaptThink)
AdaptThink (Zhang et al., 2025)	40K @ 16K	40K @ 16K, K=16, T_{upd}≈16K	174,080	348,160	522,240	1.00×
ThinkLess (Fang et al., 2025)	334K @ 24K	1K @ 24K, K=8, T_{upd}≈24K	131,328	262,656	393,984	0.75×
RPO (Ours)	8K @ 16K	1K @ 16K, K=16, T_{upd}=1	6,144	4,097	10,241	0.02×

Table 5: PFLOPs accounting for an 8B dense Transformer (full-length sequences at the stated context). We assume $2P$ FLOPs/token for forward and $4P$ for backward. For AdaptThink, the number of dataset is set to 40K as reported.

Model	Stage	LiveCodeBench			GPQA-Diamond			Avg			
		P@1	#Tok	Think%	P@1	#Tok	Think%	P@1	#Tok	Think%	RAR↓
Exaone4-1.2B	Baseline	42.2	12,159	100.0	50.8	7,145	100.0	46.5	9,652	100.0	2.15
	+ SFT	34.6	9,281	90.1	46.2	5,037	70.1	40.4	7,159	80.1	1.98
	+ **RPO**	36.7	6,692	61.7	41.7	2,881	35.0	39.2	4,787	48.4	1.26
Qwen3-8B	Baseline	49.5	14,878	100.0	58.8	7,355	100.0	54.2	11,117	100.0	1.85
	+ SFT	30.2	10,410	63.0	31.1	5,227	50.1	30.7	7,819	56.6	1.85
	+ **RPO**	35.4	11,532	54.7	55.4	6,910	67.2	45.4	9,221	61.0	1.34
Qwen3-32B †	Baseline	59.1	13,498	100.0	66.2	5,108	100.0	62.7	9,303	100.0	1.60
	+ SFT	58.3	11,457	90.6	66.3	4,822	98.5	62.3	8,140	94.6	1.52
	+ **RPO**	54.4	10,623	87.2	65.0	4,768	97.6	59.7	7,696	92.4	1.55

Table 6: LiveCodeBench/GPQA-Diamond results. Models without † are trained with full-parameter SFT; † indicates a LoRA-tuned model.



(a) MATH500. (b) AIME25.

Figure 6: Reliability diagrams (Guo et al., 2017) for the root confidence $p_{\text{short}}(x)=p_\theta(z_S|x,\texttt{<think>})$. Each bin plots empirical accuracy against predicted confidence, with the diagonal indicating perfect calibration.

to correlate with correctness, but the signal is weak and dataset dependent. On AIME25, which is substantially harder, mass concentrates at low $p_{\text{short}}(\cdot)$, so the router rarely assigns high SHORT confidence, whereas MATH500 shows a broader spread with more high confidence bins. Further discussions are in Appendix.

6 Conclusion

We introduce **RPO**, a root token policy optimization framework for adaptive thinking that updates only the initial gating token while treating the remainder of the rollout as environment output. Precisely, **RPO** turns static LONG style generation into a lightweight routing problem, reducing significant training overhead during GRPO. Across multiple model families and scales, **RPO** improves the accuracy routing trade-off under most scenarios, achieving lower reasoning per accuracy ratios over expert scientific reasoning and coding as well as math reasoning. Some configurations trace a comparable but not strictly dominating frontier, supporting the use of **RPO** as a controllable routing layer rather than a uniform replacement for SFT.

Limitations

While our work supports diverse LRM families and scales, several limitations suggest avenues for future exploration. First, broader evaluation is needed under advanced benchmark datasets requiring agentic behaviors with tools. Second, while root confidence is informative under our branching problem setting, it can be not only sensitive to bucket composition (i.e., easy and hard queries for **RPO**), but weakly calibrated towards its query difficulty. Third, we focus on a binary routing decision and do not optimize intra mode behaviors such as how long to think or when to stop, which could further improve compute allocation. Fourth, **RPO**'s gain over the SFT operating point is configuration-dependent: on Qwen3-1.7B, the post-SFT routing already saturates the achievable RAR, so **RPO** yields a comparable rather than strictly better operating point; on Qwen3-32B, where we use LoRA tuning and SFT already produces a near-baseline conservative router, **RPO** likewise traces a comparable but not strictly dominating frontier. We therefore position **RPO** as a controllable routing layer to be selected via checkpoint or threshold, rather than a uniform replacement for SFT. Finally, due to limited compute, we do not provide a more exhaustive ablation of the RL training dynamics regarding policy optimization to uncover how **RPO** enables such gating behaviors in model aspect or a fully controlled comparison against alternative GRPO variants based on their publicly released checkpoints and exact training recipes. While mitigating such behaviors would be valuable, it is beyond the scope of this study. We leave other discussion points including the design of root token choice, and generalizability in Appendix.

Acknowledgments

We thank Dahyun Lee for providing extensive feedback on our paper.

References

Anthropic. 2024. Claude 3.5 Sonnet Model Card Addendum. Technical report.

Daman Arora and Andrea Zanette. 2025. Training language models to reason efficiently. *arXiv preprint arXiv:2502.04463*.

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, and 1 others. 2025. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. *arXiv preprint arXiv:2507.10524*.

Gongfan Fang, Xinyin Ma, and Xinchao Wang. 2025. Thinkless: Llm learns when to think. *arXiv preprint arXiv:2505.13379*.

Sicheng Feng, Gongfan Fang, Xinyin Ma, and Xinchao Wang. 2025. Efficient reasoning models: A survey. *arXiv preprint arXiv:2504.10903*.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. 2017. On calibration of modern neural networks. In *International conference on machine learning*, pages 1321–1330. PMLR.

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*.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. 2024. Token-budget-aware llm reasoning. *arXiv preprint arXiv:2412.18547*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Jian Hu, Xibin Wu, Zilin Zhu, Xianyu, Weixun Wang, Dehao Zhang, and Yu Cao. 2024. Openrlhf: An easy-to-use, scalable and high-performance rlhf framework. *arXiv preprint arXiv:2405.11143*.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, and 1 others. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

MAA. 2024. American invitational mathematics examination (aime). https://www.maa.org/math-competitions/aime. Accessed: 2025-09-18.

MAA. 2025. American invitational mathematics examination (aime). https://www.maa.org/math-competitions/aime. Accessed: 2025-09-18.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*.

Open-R1. 2025. OpenR1-Math-220k. Hugging Face Datasets. Accessed: 2026-01-06.

OpenAI. 2025. Gpt-5 system card. Technical report, OpenAI. Technical Report.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. 2024. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*.

LG Research, Kyunghoon Bae, Eunbi Choi, Kibong Choi, Stanley Jungkyu Choi, Yemuk Choi, Kyubeen Han, Seokhee Hong, Junwon Hwang, Taewan Hwang, and 1 others. 2025. Exaone 4.0: Unified large language models integrating non-reasoning and reasoning modes. *arXiv preprint arXiv:2507.11407*.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler. 2022. Confident adaptive language modeling. *Advances in Neural Information Processing Systems*, 35:17456–17472.

Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Shaochen Zhong, Hanjie Chen, and 1 others. 2025. Stop overthinking: A survey on efficient reasoning for large language models. *arXiv preprint arXiv:2503.16419*.

Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, Kai Dang, Xionghui Chen, Jianxin Yang, Zhenru Zhang, Yuqiong Liu, An Yang, Andrew Zhao, Yang Yue, Shiji Song, Bowen Yu, Gao Huang, and Junyang Lin. 2025a. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for llm reasoning. *arXiv preprint arXiv:2506.01939*.

Xiangqi Wang, Yue Huang, Yanbo Wang, Xiaonan Luo, Kehan Guo, Yujun Zhou, and Xiangliang Zhang. 2025b. Adareasoner: Adaptive reasoning enables more flexible thinking. *arXiv preprint arXiv:2505.17312*.

Siye Wu, Jian Xie, Yikai Zhang, Aili Chen, Kai Zhang, Yu Su, and Yanghua Xiao. 2025. Arm: Adaptive reasoning model. *arXiv preprint arXiv:2505.20258*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D Goodman. 2024.

Quiet-star: Language models can teach themselves to think before speaking. *arXiv preprint arXiv:2403.09629*.

Jiajie Zhang, Nianyi Lin, Lei Hou, Ling Feng, and Juanzi Li. 2025. AdaptThink: Reasoning Models Can Learn When to Think. *arXiv preprint arXiv:2505.13417*.

A Discussions

A.1 Root-level gating is tokenization agnostic

We demonstrate that the design of root token in **RPO** framework is tokenization agnostic over a broader range of model families. Other than the settings in the body, we observe that models without specific routing tokens can achieve robust mode routing by prepending arbitrary prefixes to distinguish between reasoning and non-reasoning modes. Qwen3 models enable mode selection through distinct single tokens for reasoning and non-reasoning modes. Interestingly, even for models lacking such routing tokens, adding arbitrary prefixes to distinguish between reasoning and non-reasoning modes enables routing while preserving base reasoning behavior. In our experiments with R1-Distill-Qwen, a reasoning-only model, we prepend a pair of instruction prefix sentences before the `<think>` in the chat template prior to SFT. Then, it is observed that the model keeps the reasoning accuracy when performing inference with the reasoning prefix, and routing get possible when no prefix is provided. Several different sentences (about 20$\sim$30 tokens) are tested and it is found that this behavior is consistent regardless of the sentence content (i.e., token sequence for prefix) (Table 8). We verify these with **RPO** training: when the prefixes are prepended during rollout and trained only on the prefix tokens, routing improved similarly as mentioned in the body of the work that utilizes only single routing tokens for training.

A.2 Alternative root token markers

A potential concern is that `\n` and `\n\n` carry semantic meaning in natural text and could introduce ambiguity. We emphasize that the routing decision is confined to a single categorical action at the root position immediately after `<think>`; newline tokens appearing later in the generation do not affect routing, and **RPO** masks gradients for all non-root tokens. Nevertheless, to verify that our method is not sensitive to the specific marker choice, we compare three marker types on Qwen3-8B (Table 8).

A.3 What generalizes and what does not

RPO often transfers when the target setting rewards *selective compute*, since the update mainly reshapes when the model spends tokens rather than teaching new domain knowledge. In practice, a router learned from verifiable signals (i.e., math problems) can still reduce LONG usage on sci-

Model	Stage	Math500			AIME24			AIME25			LiveCodeBench			GPQA		
		P@1	#Tok	Think%	P@1	#Tok	Think%	P@1	#Tok	Think%	P@1	#Tok	Think%	P@1	#Tok	Think%
Qwen3-8B	Baseline (think)	96.9	5,547	100.0	75.8	15,602	100.0	68.3	18,419	100.0	49.5	14,879	100.0	58.8	5,838	100.0
	+ SFT	90.9	3,251	45.8	50.4	11,196	50.0	44.4	11,593	51.3	34.4	10,519	54.4	56.6	7,092	96.8
	+ **RPO**	88.0	2,412	27.8	45.2	9,437	38.1	40.6	9,650	39.6	28.7	8,362	40.1	57.1	6,908	96.2
Qwen3-32B	Baseline (think)	97.3	4,806	100.0	79.4	13,235	100.0	69.8	16,431	100.0	59.1	13,498	100.0	66.2	5,108	100.0
	+ SFT	92.1	2,602	42.7	59.8	9,309	56.3	54.2	10,379	58.3	58.3	11,457	90.6	66.3	4,822	98.5
	+ **RPO**	92.1	1,975	20.4	63.1	10,291	61.5	58.1	11,911	68.1	54.4	10,623	87.2	65.0	4,768	97.6

Table 7: LoRA-based mode control and **RPO** on Qwen3. For each model size, we compare the <think>-forced baseline (LONG) with LoRA SFT for mode selection and a subsequent **RPO** update. We report Pass@1 (%), average generated tokens, and THINK% (fraction routed to LONG). We set batch size 64 and learning rate 2×10^{-7}, and use the Qwen default prompt for math tasks. This table reports a *LoRA-tuned* variant of Qwen3-8B/32B and is therefore not directly comparable to the full-parameter SFT Qwen3-8B in Tab. 6; baseline numerical drift across the two tables reflects independent decoding runs under matched settings.

Marker type	**MathAvg P@1**	**Think%**	**LCB P@1**	**GPQA P@1**
Newline fork (default)	62.0	44.8	35.4	55.4
Prefix sentence A	61.8	45.1	35.1	55.2
Prefix sentence B	61.5	46.3	34.8	54.9

Table 8: Comparison of root token marker types on Qwen3-8B after +**RPO**. *Newline fork* uses \n vs. \n\n. *Prefix sentence A/B* use distinct instructional prefixes (∼20–30 tokens) prepended before <think> to distinguish modes.

entific and coding benchmarks, but accuracy improvements are not guaranteed because the optimal routing rule is dataset dependent (see the heterogeneous Think%/P@1 trade-offs across LiveCodeBench and GPQA-Diamond in Table 6). Calibration does not transfer perfectly either. Root confidence is informative but can be weakly correlated on hard suites where p_{short} concentrates near low values. While **RPO** adaptively selects the mode depending on the learned query difficulty, in practice, deploying such way can be brittle across real-world scenarios. In this situation, we may need to treat checkpoint or threshold selection as *dataset-conditioned calibration* in case-by-case, since similar accuracy can coincide with materially different thinking ratios.

B Implementation details

B.1 Hyperparameter optimization for SFT

Through fine-tuning R1-Distill, Qwen3, and EXAONE4 models on math benchmarks, we observed that preserving the both accuracy and number of tokens of original model requires different hyperparameter configurations depending on model family and size. For instance, R1-Distill-1.5B maintained accuracy with learning rate 5e-5, while accuracy degraded at other values; in contrast, R1-Distill-7B required learning rate 1e-5 to preserve performance.

When applying SFT with self-distilled data on Qwen3 models, we observed distinct behaviors across model sizes and modes. In reasoning mode, the 1.7B model exhibited a tendency to generate longer sequences, whereas the 8B model maintained the original token count. In non-reasoning mode, both models generally produced longer outputs; however, distilling with presence penalty 1.5 successfully maintained token counts comparable to the original models. For EXAONE4, using a lower temperature of 0.4 in non-reasoning mode preserved both accuracy and token efficiency similar to the base model.

B.2 Mode-specific GRPO rollout strategies

For Qwen3 models, the non-reasoning mode prefix (\n\n) and reasoning mode prefix (\n) each correspond to a single distinct token. When generating n rollout samples, we prepended each token to $n/2$ sample prompts before generation. For EXAONE4, the non-reasoning mode prefix corresponds to two consecutive reasoning mode tokens. Unlike Qwen3, we prepended the reasoning mode prefix to all rollout samples initially. For $n/2$ samples, we appended an additional token to enforce non-reasoning generation, and for the remaining samples, we enforced reasoning mode generation by masking the logits of the initial reasoning mode prefix token during sampling.

B.3 SFT Dataset and training

For SFT training, we used OpenR1-Math prompts to generate SFT dataset. We employed different strategies depending on model capabilities. For Qwen3 and EXAONE4 models, we applied self-distillation to generate both reasoning and non-reasoning trajectories. For Qwen3, we used tem-

Backbone	Checkpoint	Mode	Math Avg		Non-math Avg	
			P@1	Think%	P@1	Think%
Qwen3-8B	Pre	forced-think (LONG)	78.6	100.0	54.2	100.0
		forced-no-think (SHORT)	43.4	0.0	34.9	0.0
	+SFT	adaptive	63.8	57.3	30.7	56.6
		forced-think (LONG)	78.1	100.0	52.0	100.0
		forced-no-think (SHORT)	43.4	0.0	34.0	0.0
	+RPO	adaptive	62.0	44.8	45.4	61.0
		forced-think (LONG)	77.3	100.0	53.8	100.0
		forced-no-think (SHORT)	42.6	0.0	35.0	0.0
Exaone4-1.2B (v2)	Pre	forced-think (LONG)	62.9	100.0	46.5	100.0
		forced-no-think (SHORT)	37.6	0.0	31.4	0.0
	+SFT	adaptive	60.0	80.1	40.4	80.1
		forced-think (LONG)	62.8	100.0	44.7	100.0
		forced-no-think (SHORT)	43.5	0.0	30.9	0.0
	+RPO	adaptive	58.0	48.4	39.2	48.4
		forced-think (LONG)	60.8	100.0	45.6	100.0
		forced-no-think (SHORT)	41.5	0.0	31.2	0.0

Table 9: **Robustness to forcing thinking vs. non-thinking after post-training.** Math Avg is the mean P@1 over MATH500/AIME24/AIME25. Non-math Avg is the mean P@1 over LiveCodeBench and GPQA-Diamond. **Think%** is the fraction routed to thinking (100% and 0% under forced modes).

perature 0.6, top-p 0.95, and top-k 20 for reasoning mode, and temperature 0.7, top-p 0.8, and top-k 20 for non-reasoning mode. For EXAONE4, we used temperature 0.4 and top-p 0.95 for both modes, with top-k 20 applied only in reasoning mode. We generated approximately 20K examples for each mode, resulting in about 40K total training samples per model. For Qwen3 V2 (LoRA), we employed n=4 sampling with temperature 0.6, top-p 0.95, top-k 20, and presence penalty 1.5 for both reasoning and non-reasoning modes. We selected the shortest correct solution from the sampled candidates to construct the SFT dataset.

For R1-1.5B and 7B, which runs exclusively in reasoning mode, we utilized external resources. We sampled the original OpenR1-Math solution for reasoning mode trajectories and used Qwen2-Math-72B-Instruct to distill corresponding non-thinking trajectories with temperature 0.7 and top-p 0.8. Additionally, since R1-Distilled-Qwen lacks specific control tokens to distinguish between reasoning and non-reasoning modes (unlike Qwen3's `\n` vs. `\n\n`), we prepended a short instructional prefixes to each mode.

B.4 Dataset sizes for LoRA updates

For our framework with LoRA updates, we use substantially fewer SFT queries than full parameter tuning. Mode selection only requires teaching the model to follow the two templates and to expose the root decision token, rather than relearning problem solving. Concretely, we draw math queries from OpenR1-Math (Open-R1, 2025), self generate candidate trajectories under each mode, and keep only correctness and template valid demonstrations, but sample only **4K per mode** (8K total) for LoRA SFT, instead of 20K per mode used in full fine tuning. We keep the **RPO** stage unchanged and run root token GRPO with the same rollout and reward settings as in the main experiments, so the reduced data primarily affects the SFT mode controllability step rather than the routing optimization.

B.5 SFT stop criterion and sensitivity

As described in 1, the SFT stage is intentionally designed as a *controllability enabler* rather than a capability-improving stage. We apply an explicit stop criterion: training continues only while the default-template accuracy remains within a tolerance ϵ of the original model. This prevents overtraining, which can degrade both overall accuracy and routing quality (e.g., high Think% but low performance due to template drift).

To illustrate sensitivity to SFT duration, we report forced-LONG accuracy at different SFT epochs on R1-Distill models in Table 10. Over-

Table 10: SFT sensitivity to training epochs on R1-Distill models under forced-LONG evaluation.

Model	Stage	MATH500 P@1	AIME24 P@1
R1-Distill-1.5B	Baseline	83.2	29.4
	SFT 1 ep	75.1	20.8
	SFT 3 ep	77.6	22.3
R1-Distill-7B	Baseline	93.5	54.2
	SFT 1 ep	90.4	43.5
	SFT 2 ep	88.0	44.5

Table 11: Budget-matched comparison on MATH500 (R1-Distill-1.5B). Under equal PFLOPs (∼10K), full-trajectory GRPO becomes under-trained and achieves only modest routing (Think%=90.2%), while **RPO** achieves comparable accuracy with substantially lower Think%. Different **RPO** checkpoints trace a Pareto frontier between accuracy and compute.

Setting	Update scope	PFLOPs	MATH500 P@1	Think%
Full GRPO (budget-matched)	all tokens	∼10K	80.8	90.2
RPO (ckpt A)	root-only	∼10K	77.6	20.5
RPO (ckpt B)	root-only	∼10K	80.4	43.4

training can degrade the solver: for R1-Distill-Qwen-1.5B, SFT at 1 epoch drops MATH500 from 83.2 to 75.1, while 3 epochs partially recovers to 77.6 but at higher training cost. For R1-Distill-Qwen-7B, 1 epoch (90.4) outperforms 2 epochs (88.0) on MATH500, confirming that the optimal SFT duration is model-dependent and motivating the tolerance-based stop rule.

C Additional results

C.1 LoRA updates

Table 7 shows LoRA-based SFT and subsequent **RPO** on Qwen3 models. LoRA SFT mainly teaches template-following and root-level mode controllability, so it already yields meaningful reductions in Think% and tokens compared to the always-<think> baseline. Building on this controllable initialization, **RPO** further shifts the root gating distribution and provides a stronger compute-accuracy tradeoff, often reducing Think% substantially while keeping task performance non-trivial across both math and non-math benchmarks. The resulting operating point is model-size dependent, but consistently shows that root-only RL can refine routing without requiring full-trajectory policy updates.

C.2 Robustness to hard forcing of thinking after post-training

Our method learns a root-level routing policy that chooses between a *thinking* and a *non-thinking* generation mode. A potential failure mode is *entanglement*: post-training might specialize the model such that performance depends on the router, or that one mode becomes unreliable when selected deterministically.

We therefore evaluate each checkpoint under two *hard forcing* interventions. In **forced-think (LONG)**, we always select the reasoning root token, requiring an explicit trace. In **forced-no-think (SHORT)**, we always select the non-reasoning root token, suppressing the trace. We report P@1, average generated tokens, and the fraction of examples routed to thinking (**Think%**).

Table 9 summarizes results on math reasoning (MATH500/AIME24/AIME25) and non-math benchmarks (LiveCodeBench/GPQA-Diamond). Across backbones and checkpoints, we find that post-trained models remain functional under both forced regimes: forcing *thinking* preserves a large portion of the always-think baseline accuracy, while forcing *non-thinking* yields non-trivial performance with substantially reduced tokens. Overall, these results indicate that SFT/**RPO**primarily reshapes the *routing distribution* (i.e., when to allocate reasoning), without collapsing either mode into a brittle specialization.

C.3 Budget-matched comparison: root-only vs. full-trajectory RL

A key question is whether the accuracy gap between **RPO** and full-trajectory methods (e.g., AdaptThink) stems from the root-only formulation itself or simply from differences in training budget. To disentangle these effects, we compare **RPO** with full-trajectory GRPO (i.e., AdaptThink) under the same PFLOPs budget on MATH500 using R1-Distill-Qwen-1.5B (Table 11).

Under equal PFLOPs, full-trajectory GRPO is under-trained and barely reduces Think% (90.2%), whereas **RPO** achieves meaningful routing (20.5–43.4% Think) with comparable or slightly lower accuracy. This confirms that root-only updates are more sample-efficient for the routing objective under low-budget regimes, which is the primary use case **RPO** targets.

C.4 Post-RPO threshold routing

To verify that inference-time controllability is preserved after **RPO** training, we sweep the routing threshold τ on the final +RPO checkpoint (Table 12). Higher τ routes more queries to LONG, in-

Table 12: Threshold routing (τ) sweep on the final **+RPO** checkpoint (Qwen3-8B). τ controls the fraction routed to LONG.

τ	Think%	MathAvg P@1	Non-mathAvg P@1
0.90	60.0	63.0	46.0
0.80	52.3	62.5	45.8
0.70	44.8	62.0	45.4
0.60	38.1	60.8	44.2

Table 13: Outcome bucket distribution on the **RPO** training set (Qwen3-8B).

Difficulty	LONG-only	SHORT-only	Both-correct	Both-wrong
Easy ($\text{acc}_S \geq 0.75$)	3.2%	1.8%	89.5%	5.5%
Medium	28.4%	4.1%	42.3%	25.2%
Hard ($\text{acc}_S < \text{acc}_L \leq 0.5$)	45.7%	2.3%	8.6%	43.4%

creasing accuracy at the cost of more tokens; lower τ saves compute with a controlled accuracy reduction. This demonstrates that τ provides a smooth, monotonic compute–accuracy trade-off even after **RPO** training, with the forced-LONG/forced-SHORT results (Table 9) serving as upper/lower capability bounds.

C.5 Automatic labeling quality

Our automatic labeling assigns each query to one of four outcome buckets based on paired evaluation under both SHORT and LONG modes: (i) *LONG-only-correct*: only LONG succeeds, (ii) *SHORT-only-correct*: only SHORT succeeds, (iii) *both-correct*: both modes succeed, and (iv) *both-wrong*: neither succeeds. This paired design directly mitigates the concern that SHORT might answer correctly by chance: such instances fall into *SHORT-only-correct* or *both-correct* and are not used as evidence that LONG is required.

Table 13 reports the bucket distribution for Qwen3-8B on the RPO training set. On hard queries (e.g., AIME-level), the *LONG-only-correct* bucket dominates, indicating a strong and consistent mode separation signal. The *SHORT-only-correct* bucket is small across all difficulty levels, suggesting that chance-correct SHORT events rarely contaminate the supervision.

To make the bucket assignments concrete, we provide representative training-set queries per bucket together with a mechanistic rationale for why each problem falls into the assigned bucket under Qwen3-8B mode-conditioned generation (Table 14). The rationales are not post-hoc rationalizations: they identify the specific reasoning step (e.g., modular case enumeration, sign disambiguation) at which SHORT mode systematically truncates, or at which LONG mode introduces a derivation tangent.

Table 14: Representative training-set queries per outcome bucket (Qwen3-8B). Queries are abbreviated; rationales identify the reasoning step that drives the bucket assignment.

Bucket	Query (abbrev.)	Mechanistic rationale
LONG-only	"Find the number of positive integers $n \leq 2024$ such that $n^2 + n + 1 \equiv 0 \pmod 7$."	Requires enumerating $n \bmod 7$ (solutions: $n \equiv 2, 4$), then counting via floor arithmetic. SHORT typically guesses a single residue or skips the residue scan and returns an off-by-one count.
SHORT-only	"Compute $\cos(\pi/3)$."	Closed-form recall ($1/2$). SHORT returns the memorized value. LONG occasionally derives via half-angle identities and conflates $\cos(\pi/6)$ with $\cos(\pi/3)$, yielding $\sqrt{3}/2$.
Both-correct	"Solve $\log_2 x = 5$ for x."	Single inverse step: $x = 2^5 = 32$. No branch points or memorization gaps; both modes converge.
Both-wrong	"Find the smallest positive integer n such that the decimal expansion of $1/n$ has period exactly 23."	Requires knowing period equals $\text{ord}_n(10)$ when $\gcd(n, 10)=1$ and that $10^{23} - 1$ admits a non-trivial prime factor. Beyond Qwen3-8B's reliable recall in either mode.

C.6 Misrouting analysis

We analyze routing errors by counterfactual evaluation: for each query where the adaptive model is incorrect, we re-evaluate under the opposite forced mode to determine whether the error is attributable to the gate decision. We categorize errors into:

- **False-SHORT** (harmful misroute): the model chose SHORT but LONG would have succeeded. These typically involve hidden multi-step constraints or subtle arithmetic that require explicit deliberation.
- **False-LONG** (wasteful misroute): the model chose LONG when SHORT would have sufficed, or when overthinking amplified distractors or hallucinated intermediate claims.
- **Both wrong**: neither mode succeeds (out-of-distribution or missing knowledge).
- **Both correct**: compute-saving opportunity, not an error.

Table 15: Misrouting composition on MATH500 (R1-Distill-1.5B, 500 queries). Lowering τ (0.75→0.70) increases the number of false-SHORT errors (+12) while only slightly reducing false-LONG errors (-4), reflecting a clear compute-risk trade-off.

τ	**False-SHORT**	**False-LONG**	**Both-wrong**
0.75	4	77	0
0.70	16	73	0

Table 16: Representative misrouting cases on MATH500 (R1-Distill-1.5B, τ=0.70). *Adaptive* reports the mode chosen by the learned router and its correctness; *Counterfactual* reports the opposite forced mode's outcome on the same query.

Category	**Query (abbrev.)**	**Adaptive**	**Counterfactual**
False-SHORT	"How many distinct arrangements are there of the letters of MISSISSIPPI?"	SHORT, ✗ (returns 11!)	LONG, ✓ ($11!/(4!\,4!\,2!) = 34{,}650$)
False-LONG	"Compute $\sum_{k=1}^{10} k$."	LONG, ✗ (drifts to $\sum k^2$ formula)	SHORT, ✓ (55)
Both-wrong	"Find the smallest n such that the decimal expansion of $1/n$ has period exactly 23."	LONG, ✗	SHORT, ✗
Both-correct	"Solve $3x + 7 = 22$ for x."	SHORT, ✓	LONG, ✓

Table 15 reports the misrouting composition on MATH500 (R1-Distill-1.5B) at two threshold settings, extracted from counterfactual re-evaluation. As shown in Table 15, false-LONG (conservative over-routing) is the dominant error type, while false-SHORT errors are relatively rare and increase predictably as τ decreases. This suggests that the learned router errs on the side of caution (spending LONG when unnecessary) rather than harmfully skipping reasoning. A plausible source of this conservative bias is the pretraining prior that correlates more tokens with trying harder. Root-level post-training can reshape the gate, but may not fully overwrite this prior, motivating future work on calibration-aware training.

To make the failure modes diagnosable rather than aggregate, we provide representative cases per category in Table 16. Each row shows the routed mode under the adaptive policy together with the counterfactual outcome under the opposite forced mode, isolating whether the error is attributable to the gate decision (false-SHORT/false-LONG) or to a solver limitation (both-wrong). The false-SHORT example illustrates a query whose surface form (a permutation count) admits a one-line answer but requires a divisor correction that SHORT systematically omits; the false-LONG example illustrates a query whose closed-form invites SHORT, while LONG introduces an unnecessary derivation chain that drifts to a sibling identity.

Table 17: LiveCodeBench pass@k results with token cost. Under pass@10 evaluation, the adaptive policy reduces token cost while maintaining competitive pass@10 performance.

Model	**Stage**	**pass@1**	**pass@10**	**Avg Tokens**
Qwen3-32B	Baseline	59.1	75.3	13,498
	+ SFT	58.3	77.1	11,457
	+ **RPO**	54.4	72.8	10,623
Exaone4-1.2B	Baseline	42.2	62.9	12,159
	+ SFT	34.6	56.0	9,281
	+ **RPO**	36.7	58.2	6,692

C.7 Pass@k evaluation on coding tasks

We additionally report pass@10 alongside token cost on LiveCodeBench (Table 17). Under the pass@k lens, the adaptive policy often matches or improves pass@10 relative to the baseline while reducing average token cost, which is the trade-off view most relevant for coding deployment where multiple samples are drawn.

D AI Usage

We utilize AI tools (OpenAI's ChatGPT and Google's Gemini) to support code generation for experiments and help refine the text in a sentence-level. The core idea of our method, design of the framework, and the majority of the manuscript are written by the authors.