TreeAdv: Tree-Structured Advantage Redistribution for Group-Based RL

Anonymous ACL submission

Abstract

Reinforcement learning with group-based objectives, such as Group Relative Policy Optimization (GRPO), is a common framework for aligning large language models on complex reasoning tasks. However, standard GRPO treats each rollout trajectory as an independent sequence and assigns one single sequence-level advantage to all tokens, which leads to sample inefficiency and a length bias toward verbose or redundant chains of thought without improving logical depth. We introduce TreeAdv (Tree-Structured Advantage Redistribution for Group-Based RL), which makes the tree structure of group rollouts explicit for both exploration and advantage assignment. Specifically, TreeAdv builds a group of trees (a forest) based on an entropy-driven sampling method where each tree branches at high-uncertainty decisions while sharing low-uncertainty tokens across rollouts. Then, TreeAdv aggregates token-level advantages for internal tree segments by redistributing the advantages of complete rollouts (all leaf nodes), and TreeAdv can easily apply to group-based objectives such as GRPO or GSPO. Across 10 math reasoning benchmarks, TreeAdv consistently outperforms GRPO and GSPO, while using substantially fewer generated tokens under identical supervision, data, and decoding budgets.

1 Introduction

Since the emergence of o1-style reasoning models (OpenAI et al., 2024; OpenAI, 2024), test-time scaling (Muennighoff et al., 2025) has received renewed attention: allocating more computation at inference—e.g., spending more time "thinking" or exploring multiple candidate reasoning trajectories—can significantly improve performance on difficult reasoning tasks (Jiang et al., 2025; Google DeepMind, 2025).

To fully benefit from test-time scaling, models must be trained to produce reliable multi-step reasoning trajectories rather than merely longer ones. Reinforcement learning (RL) post-training is an effective mechanism for eliciting reliable multi-step reasoning in large language models (LLMs), especially for mathematics problem solving (Zhang et al., 2025; Phan et al., 2025). Group-based policy optimization methods such as GRPO (Shao et al., 2024) and GSPO (Zheng et al., 2025) are increasingly adopted as practical *PPO-style* recipes for scaling reasoning RL: they use group statistics as baselines to avoid training a separate value network, reducing memory overhead while retaining PPO-like policy updates (Yang et al., 2025a; DeepSeek-AI et al., 2025).

However, GRPO-style training largely relies on *sequence-level credit assignment* (Li et al., 2025a), where each sampled response y receives a terminal scalar reward, which is converted into a single advantage A and applied uniformly to all tokens of y. This can reinforce redundant or locally suboptimal reasoning whenever the final answer is correct and yields noisy updates because key decisions, corrected mistakes, and detours share the same scalar supervision (Ye et al., 2025; Zhang et al., 2025).

We address this limitation without dense process supervision or additional reward models. Instead, we exploit the shared-prefix structure among multiple sampled continuations from the same prompt, where these continuations naturally form a tree of branched rollouts. By aggregating terminal results over this topology, we derive structure-aware segment or token advantages for intermediate decisions (Yang et al., 2025b).

We propose Tree-Structured Advantage Redistribution for Group-Based RL (TreeAdv), a drop-in modification to PPO-style group-based RL (GRPO/GSPO). Empirically, TreeAdv improves long-form reasoning performance and reduces the average generated tokens under matched rollout budgets and decoding settings. In more detail, TreeAdv calculate token-level advantages by reorganizing rollout trajectories which are sampled

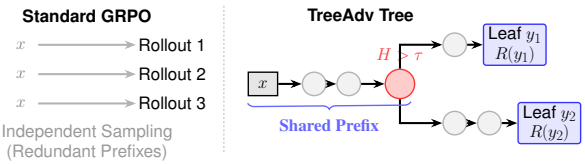

Figure 1: Comparison of trajectory generation. **Left:** Standard GRPO samples independent rollouts that redundantly repeat prefixes. **Right:** TreeAdv constructs a tree by sharing prefixes and branching only at high-entropy states ($H > \tau$).

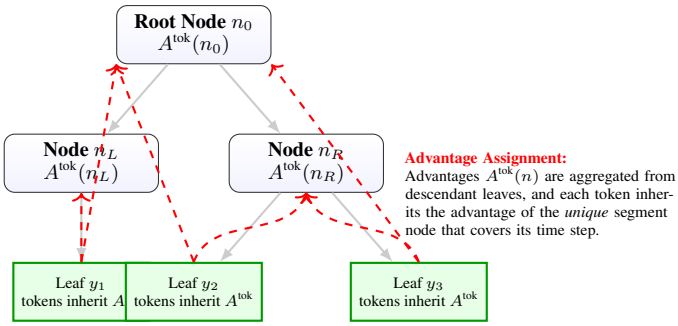

Figure 2: Illustration of TreeAdv's tree-aware advantage construction. Scalar rewards are computed only on complete rollouts (leaves) and converted to sequence-level advantages. These leaf advantages are aggregated into segment-level advantages $A^{\text{tok}}(n)$. Each token then receives the advantage of the unique segment node that covers its time step.

by a designed entropy based branching strategy. And then TreeAdv utilize these token-level advantages for any group based RL methods, such as GRPO and GSPO. We demonstrate our method over commonly used math reasoning benchmarks and compared with sota baselines, where our method consistently outperforms them in both accuracy and token-consumption.

Our contributions are: (i) topology-aware advantage redistribution for GRPO/GSPO with minimal pipeline changes; (ii) improved accuracy and shorter reasoning on long-form math benchmarks; (iii) analysis of sequence-level credit assignment failure modes and their mitigation.

2 Related Work

2.1 Group-based Policy Optimization

PPO-style policy optimization (Schulman et al., 2017) with GAE-style estimators (Schulman et al., 2016) is effective but often costly for LLM post-training due to the value network. Group-based policy optimization (Chen et al., 2025; Nimmaturi et al., 2025) replaces learned baselines with group statistics, enabling PPO-style updates without training a separate critic/value network. GRPO (Shao et al., 2024) computes group-normalized advantages from sampled rollouts, and GSPO (Zheng et al., 2025) improves stability via sequence-level smoothing/weighting while preserving the same PPO-style training backbone (Yang et al., 2025a). A shared limitation is sequence-level supervision: outcomes are defined on complete rollouts, making it difficult to assign distinct credit to intermediate reasoning segments (Li et al., 2025a; Ye et al., 2025).

2.2 Tree-based Reasoning and Rollout Strategies

Tree-structured search improves inference-time reasoning by exploring branched continuations, as in ToT (Yao et al., 2023) and MCTS-style methods (Ding et al., 2024), with strong results on challenging benchmarks (Wang et al., 2025; Zhang et al., 2024). For training, TreeRL (Hou et al., 2025) derives step-level advantages via Monte Carlo value/return estimation over descendant outcomes, while TreeRPO (Yang et al., 2025b) performs local sibling normalization at each step. TreePO (Li et al., 2025b) adopts DAPO (Yu et al., 2025) for process preference optimization, which in their setting requires additional rollouts (or extra sampled candidates) to construct per-step preference comparisons, along with extra per-step scoring and bookkeeping. Differently, our method keeps supervision terminal and critic-free, and uses shared-prefix topology to redistribute terminal group-relative advantages to internal segments via subtree-based aggregation(See figure 2).

3 Method

3.1 Motivation

A recurring challenge in RL post-training for multi-step reasoning is that not all tokens are equally decision-critical (Wang and Zhou, 2024). In practice, models often exhibit low uncertainty for most local steps, while a small number of positions can concentrate higher ambiguity: different plausible next tokens can lead to substantially different downstream reasoning trajectories (Ma et al., 2025). This suggests that both exploration and credit assignment should focus on these high-uncertainty positions, rather than treating every token in a rollout as equally informative.

Prior work has explored allocating branching

or process-level effort based on token uncertainty (e.g., entropy) (Li et al., 2025b; Cao et al., 2026). The key intuition is simple: branching at high-entropy positions produces rollouts that differ in meaningful local decisions under the same sampling budget, making the resulting outcomes more informative for learning.

These observations naturally connect to group-based RL. If we branch at uncertain prefixes, the sampled continuations form a shared-prefix tree, where leaves correspond to complete rollouts with terminal rewards. This structure enables prefix-conditioned comparisons among alternative local decisions made at the same prefix. TreeAdv leverages this topology to redistribute terminal, group-normalized advantages from leaf rollouts back to the corresponding shared-prefix segments, yielding token-/segment-level learning signals while keeping the standard clipped surrogate objective unchanged.

3.2 Background: Group-Based RL

For a prompt x, we sample a group of K rollout trajectories, where the i-th trajectory at step t is denoted as $s_{i,t} = (x, a_{i,1}, \cdots, a_{i,t})$ from a policy $\pi_\theta(a_{i,t} \mid s_{i,<t})$. Group-based objectives (e.g., GRPO) rely solely on a scalar sequence-level reward per rollout, R_i, and form a group-normalized sequence advantage without a value network:

$$A_i = \frac{R_i - \mu_R}{\sigma_R}, \tag{1}$$

where $\mu_R = \frac{1}{K}\sum_{j=1}^{K} R_j$, and $\sigma_R = \sqrt{\frac{1}{K}\sum_{j=1}^{K}(R_j - \mu_R)^2 + \delta}$.

In the following, we take GRPO as an example of applying a token-level optimization using our TreeAdv. Let π_{old} denote the behavior policy for generating rollouts. GRPO optimizes a PPO style clipped surrogate with per-token likelihood ratios $r_{i,t}(\theta) = \frac{\pi_\theta(a_{i,t}|s_{i,<t})}{\pi_{\text{old}}(a_{i,t}|s_{i,<t})}$, and $\bar{r}_{i,t}(\theta) = \text{clip}(r_{i,t}(\theta), 1-\epsilon, 1+\epsilon)$, and the final objective $\mathcal{L}_{\text{GRPO}}(\theta)$ is defined as

$$\mathbb{E}\left[\frac{1}{K}\sum_{i=1}^{K}\sum_{t=1}^{|s_i|}\min(r_{i,t}(\theta)A_i, \bar{r}_{i,t}(\theta)A_i)\right] \tag{2}$$

In this setup, the advantage score A_i is applied at the sequence level and to all tokens within the rollout s_i.

3.3 TreeAdv: Entropy-Guided Trees and Token-Level Advantage

Recent studies show that tokens vary in their importance and should consequently receive distinct advantage scores (Wang et al., 2024; Yang et al., 2025b). Empirically, we observed that most of the reasoning steps generated by current RL models are correct, while a few intermediates of them make inference errors. These critical reasoning steps should be reinforced more, while standard group-based RL fails to distinguish them, leading to sample inefficiency and a tendency to generate unnecessarily long rollouts.

Motivated by these observations, we proposed a method, TreeAdv, which considers token-level advantages. Specifically, our method samples rollout trees (rather than independent sequences) using an entropy-driven branching strategy. Then, TreeAdv converts leaf (complete-rollout) advantages into token-level advantages by organizing the K rollouts of each prompt into a prefix forest and aggregating advantages on shared prefix segments.

Entropy-guided branching. We measure uncertainty at a prefix state s_t, omitting the index i, by the token entropy under the behavior policy:

$$H(s_t) = -\sum_{a\in\mathcal{V}} \pi_{\text{old}}(a_t \mid s_{<t}) \log \pi_{\text{old}}(a_t \mid s_{<t}).$$

Starting from the prompt, we perform linear rollouts under π_{old}. Positions where entropy $H(s_t)$ exceeds a threshold τ are treated as *branching points*. At these points, we sample multiple distinct next tokens to create child nodes, expanding the frontier (Appendix A.1). This process yields (i) a set of completed rollouts (leaves) for reward evaluation, and (ii) internal nodes representing shared reasoning segments.

Assigning token advantages For each rollout trajectory s_i, we compute the standard group-normalized sequence advantages A_i as in Eq. 1. The token advantage $A_{i,t}$ of token $a_{i,t}$ is defined by summing up the advantage scores of the trajectory set $S(a_{i,t})$, consisting of all trajectories sharing $a_{i,t}$, and then normalized by the number of trajectories,

$$A_{i,t}^{\text{tok}} = \frac{1}{|S(a_{i,t})|}\sum_{\ell\in S(a_{i,t})} A_\ell. \tag{3}$$

The normalize operation can prevent advantages near the root (with many descendants) from dominating by scale.

TreeAdv objective in GRPO TreeAdv integrates $\tilde{A}^{\text{tok}}_{i,t}$ into the same clipped surrogate as GRPO, the objective $\mathcal{L}_{\text{TreeAdv}}(\theta)$ is defined as,

$$\mathbb{E}\left[\frac{1}{K}\sum_{i=1}^{K}\sum_{t=1}^{|s_i|}\min\Big(r_{i,t}(\theta)\tilde{A}^{\text{tok}}_{i,t}, \bar{r}_{i,t}(\theta)\tilde{A}^{\text{tok}}_{i,t}\Big)\right] \tag{4}$$

All other components, e.g., KL regularization to a reference policy, remain unchanged, and TreeAdv only modifies the construction of advantages on the token level.

TreeAdv objective in GSPO TreeAdv integrates $\tilde{A}^{\text{tok}}_{i,t}$ into the same clipped surrogate form, while replacing a length-normalized sequence-level importance ratio (PPO-style) with the per-token likelihood ratio. We follow the definition of GSPO-token and define the length-normalized sequence ratio for trajectory i as

$$w_i(\theta) = \exp\left(\frac{1}{|s_i|}\sum_{t=1}^{|s_i|}\log\frac{\pi_\theta(a_{i,t} \mid s_{i,<t})}{\pi_{\text{old}}(a_{i,t} \mid s_{i,<t})}\right) \tag{5}$$

and construct a token-wise ratio

$$w_{i,t}(\theta) = \text{sg}[w_i(\theta)] \cdot \frac{\pi_\theta(a_{i,t} \mid s_{i,<t})}{\text{sg}[\pi_\theta(a_{i,t} \mid s_{i,<t})]}, \tag{6}$$

where $\text{sg}[\cdot]$ denotes the stop-gradient operator. We further define the clipped ratio $\bar{w}_{i,t}(\theta) = \text{clip}(w_{i,t}(\theta), 1-\epsilon, 1+\epsilon)$. Then, the objective $\mathcal{L}_{\text{TreeAdv}}(\theta)$ for GSPO is defined as

$$\mathbb{E}\left[\frac{1}{K}\sum_{i=1}^{K}\sum_{t=1}^{|s_i|}\min\Big(w_{i,t}(\theta)\tilde{A}^{\text{tok}}_{i,t}, \bar{w}_{i,t}(\theta)\tilde{A}^{\text{tok}}_{i,t}\Big)\right]. \tag{7}$$

Similarly, TreeAdv only modifies the construction of advantages on the token level while adopting a GSPO-style sequence ratio for clipping and optimization.

4 Experiments

4.1 Experimental Setup

Models We conduct our primary evaluations using three base models from the Qwen3 family (Yang et al., 2025a), selected to span different scales and architectures: `Qwen3-4B-Base/Inst/2507`, `Qwen3-8B-Base/Inst`, and a Mixture-of-Experts (MoE) model `Qwen3-30B-A3B-2507`.

To contextualize our results within the state-of-the-art models, we also report performance for a set of strong open-weight reference models, including Qwen3 families (8B, 30B) and the distilled DeepSeek-R1 series (Qwen-based 7B/14B/32B and Llama-based 8B (DeepSeek-AI et al., 2025)). Note that these reference models are evaluated "as-is" without further training via TreeAdv.

Benchmarks and Metrics Our evaluation spans 10 benchmarks, grouped into three domains: (1) *Standard Mathematics*: MATH500 (Hendrycks et al., 2021) serves as the primary baseline. (2) *Olympiad-Level Competitions*: We use Omni-Math (Gao et al., 2025) for broad coverage, and a suite of high-difficulty datasets including AIME 2024/2025 (Mathematical Association of America, 2024, 2025), HMMT 2025 (Balunović et al., 2025), BRUMO 2025 (Brown University Math Olympiad, 2025), CMI (CMINMC 2025) (Carnegie Mellon Informatics and Mathematics Competition, 2025), and the OlymMATH-Easy/Hard subsets (Sun et al., 2025). (3) *Scientific Reasoning*: We use GPQA-Diamond (Rein et al., 2023) to assess graduate-level scientific problem-solving.

We report accuracy using Pass@1 for large-scale benchmarks. For smaller, high-variance datasets (at most 40 problems, e.g., AIME, HMMT), we report Pass@32 to ensure statistical stability. Additionally, we measure reasoning efficiency (**Tok**) using the average number of tokens generated per solution, where a lower count indicates more concise reasoning.

Baselines and Training Details We compare against two group-based RL baselines, GRPO and GSPO, and apply TreeAdv on top of both. Given that standard GRPO often exhibits convergence instability on Mixture-of-Experts (MoE) architectures, we exclusively employ the more robust GSPO baseline for the `Qwen3-30B-Instruct` MoE model.

In addition to these primary baselines, we further compare TreeAdv with two representative strong alternatives that incorporate step-/tree-structured training signals: DAPO (Yu et al., 2025) and TreeRL (Hou et al., 2025). These methods differ from GRPO/GSPO-style training in their supervision granularity and/or training-time computation. We therefore report them as complementary baselines and explicitly match training settings where applicable, while also disclosing any deviations in rollout budget or extra scoring required by the

Method	AIME24 (32p1)	AIME25 (32p1)	BRUMO (32p1)	CMI (32p1)	GPQA	HMMT (32p1)	MATH	OlymE	OlymH	Omni	Avg ↑ (%)	Tok ↓ (#)
(A) SOTA open models (DeepSeek)												
`DeepSeek-R1-Distill-Qwen-7B`	53.23	38.96	51.67	28.98	23.74	24.48	91.60	42.00	14.00	44.26	41.29	12222
`DeepSeek-R1-Distill-Qwen-14B`	69.90	48.75	60.21	36.80	41.92	33.33	92.80	51.00	17.00	49.55	50.13	12118
`DeepSeek-R1-Distill-Qwen-32B`	69.06	54.38	64.38	40.55	40.91	33.33	94.00	64.00	13.00	51.60	52.52	11603
`DeepSeek-R1-Distill-Llama-8B`	41.98	30.31	38.02	19.84	29.29	21.88	81.00	29.00	7.00	38.03	33.64	12854
(B) Our TreeAdv on base models												
`Qwen3-8B-Base`	11.25	9.48	18.85	4.45	18.69	1.35	68.20	7.00	3.00	24.07	16.64	2667
`w. GRPO`	30.00	22.00	37.00	16.00	48.00	12.00	88.00	17.00	4.00	36.00	31.00	7025
`w. TreeAdv-GRPO`	27.71	24.79	37.19	16.09	49.49	11.77	85.60	20.00	10.00	34.03	31.67	6171
`w. GSPO`	22.08	17.40	32.40	11.33	41.41	9.38	84.40	6.00	2.00	31.91	25.83	9583
`w. TreeAdv-GSPO`	21.98	16.35	29.79	11.41	42.42	8.02	85.00	11.00	6.00	30.71	26.27	8319
`Qwen3-4B-Base`	9.58	8.13	18.33	3.52	24.75	1.35	71.60	6.00	1.00	24.57	16.88	2628
`w. GRPO`	21.88	13.33	24.69	8.05	40.91	5.00	79.60	10.00	1.00	28.70	23.32	2794
`w. TreeAdv-GRPO`	21.25	19.17	23.02	12.11	45.45	8.85	84.00	15.00	3.00	32.05	26.39	5894
`w. GSPO`	14.58	10.52	21.15	5.31	42.42	1.46	78.00	9.00	2.00	25.43	20.99	6346
`w. TreeAdv-GSPO`	13.44	12.29	19.48	7.03	40.91	3.13	78.60	9.00	4.00	25.34	21.32	8606
(C) Our TreeAdv on instruct models												
`Qwen3-4B-Inst Think`	71.98	64.90	63.23	43.91	54.04	42.08	94.20	67.00	20.00	53.16	57.45	15322
`w. GRPO`	70.94	62.50	60.83	40.47	54.04	38.02	94.40	68.00	16.00	52.30	55.75	15000
`w. TreeAdv-GRPO`	71.88	63.96	61.15	43.36	53.03	41.25	93.40	67.00	19.00	52.55	56.66	13439
`Qwen3-4B-Inst Non-Think`	22.08	18.65	31.15	15.94	40.91	11.04	83.40	16.00	4.00	33.83	27.70	3333
`w. GRPO`	46.04	34.48	47.92	20.70	48.48	21.15	89.00	36.00	9.00	43.29	39.61	9710
`w. TreeAdv-GRPO`	45.00	39.69	47.81	25.86	49.49	25.21	89.60	37.00	9.00	43.70	41.24	8015
`Qwen3-8B-Inst Think`	74.58	66.35	67.40	45.47	61.62	40.94	93.40	74.00	18.00	54.22	59.60	16200
`w. GRPO`	75.21	66.77	67.71	45.16	58.08	43.96	93.60	**78.00**	23.00	53.97	60.55	15693
`w. TreeAdv-GRPO`	75.83	**68.13**	69.27	46.64	61.62	**45.31**	93.80	**78.00**	**27.00**	54.34	**61.99**	12073
`Qwen3-8B-Inst Non-Think`	28.02	21.15	30.94	14.69	44.44	10.52	83.20	23.00	4.00	35.41	29.54	3506
`w. GRPO`	56.25	40.00	52.92	30.31	59.60	26.98	90.80	44.00	11.00	50.52	46.24	13373
`w. TreeAdv-GRPO`	60.63	49.17	55.00	33.44	54.55	29.90	92.20	58.00	12.00	48.83	49.37	9962
`w. GSPO`	58.02	46.98	55.10	28.13	55.05	28.96	93.00	48.00	11.00	48.13	47.24	9420
`w. TreeAdv-GSPO`	60.83	48.33	55.21	30.63	57.58	29.58	91.60	46.00	14.00	47.74	48.15	11750
`Qwen3-4B-Inst-2507`	61.56	45.73	57.81	32.03	60.10	30.31	93.40	57.00	11.00	49.55	49.85	7701
`w. GRPO`	60.73	46.25	54.58	30.86	60.61	29.69	93.20	54.00	9.00	48.40	48.73	10397
`w. TreeAdv-GRPO`	61.98	52.81	56.98	32.34	64.14	31.67	93.80	53.00	18.00	50.72	51.54	5805
`w. GSPO`	60.10	45.10	54.90	31.56	60.10	29.69	92.69	59.00	11.00	50.34	49.44	6332
`w. TreeAdv-GSPO`	63.44	53.65	58.85	32.34	63.13	33.85	93.60	54.00	13.00	50.09	51.60	6197
`Qwen3-30B-A3B-2507`	74.48	61.15	**72.08**	44.38	67.17	44.48	94.60	73.00	20.00	**55.96**	60.73	6291
`w. GSPO`	63.00	60.63	68.02	45.08	**71.72**	42.60	95.60	76.00	25.00	55.44	60.31	5633
`w. TreeAdv-GSPO`	**75.63**	62.71	68.90	**47.11**	68.18	43.54	**95.60**	76.00	24.00	55.76	61.75	5748

Table 1: Results on a suite of reasoning benchmarks. (A) reports representative open SOTA models (DeepSeek family). (B) reports our TreeAdv on base models. (C) evaluates our TreeAdv on instruct models, using GRPO/GSPO as the corresponding baselines. For all TreeAdv entries in this table, we allocate a fixed rollout budget of K=16 into M=4 entropy-guided trees. Tok is the average number of generated tokens per solution (lower is better). Green indicates better results compared with GRPO/GSPO. Red indicates worse results compared with GRPO/GSPO. For accuracy metrics, **Bold** indicates the highest value in each column

method.

To ensure fair comparisons in our primary setting, all GRPO/GSPO and TreeAdv variants are trained on the same 10k subset sampled from DeepMath103K (He et al., 2025) under the VERL (Sheng et al., 2025) framework, with identical optimizer settings, rollout budgets, and decoding configurations. Complete hyperparameters and implementation details are provided in Appendix A.2.

4.2 Main Results

Table 1 summarizes results on our olympiad-level reasoning suite. We report average accuracy and **Tok** (average generated tokens per solution); per-benchmark token statistics are in Appendix A.4.

Across Blocks (B)–(C), TreeAdv consistently improves the accuracy–efficiency trade-off. In most settings, it increases average accuracy while reducing **Tok**. For `Qwen3-8B-Inst Think`, TreeAdv-GRPO improves average accuracy from 60.55% to 61.99% with a ∼23% reduction in **Tok** (15693 → 12073), and improves OlymH from 23%

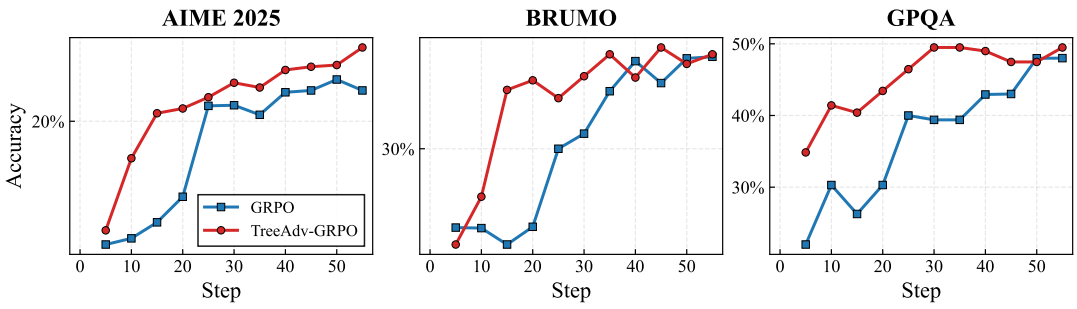

Figure 3: Test-set accuracy versus training steps on AIME25, BRUMO, and GPQA. All runs are initialized from **Qwen3-8B-Base** and compare **GRPO** against **TreeAdv-GRPO**, illustrating how accuracy evolves during training under the two objectives.

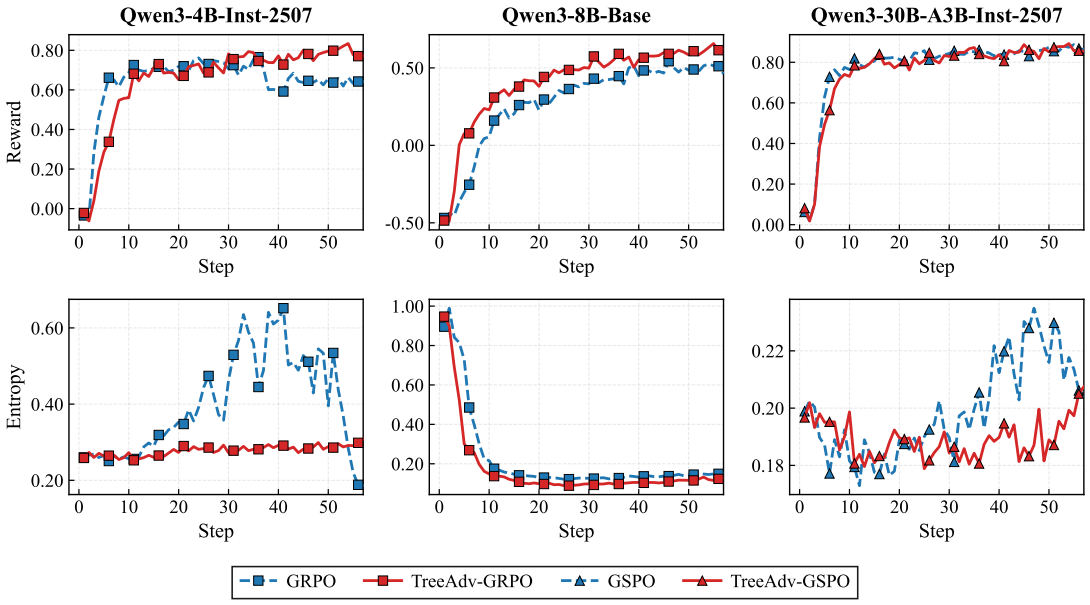

Figure 4: Training and exploration dynamics across model scales. We compare TreeAdv (Ours, red curves) with standard GRPO/GSPO baselines (blue curves) on three model sizes: Qwen3-4B-Instruct-2507, Qwen3-8B-Base, and Qwen3-30B-Instruct-2507. The top row shows reward trajectories, where TreeAdv consistently achieves higher rewards and faster convergence across scales. The bottom row shows entropy trajectories, characterizing how each method maintains exploration while optimizing the policy throughout training.

to 27%. We note that the gains on some models in Think mode can be modest, likely because the difficulty of DeepMath103K limits additional headroom under terminal-reward supervision.

Table 2 further compares TreeAdv with DAPO and TreeRL under matched supervision and decoding settings. Notably, DAPO underperforms TreeAdv even with a larger rollout budget (Appendix A.2), yielding lower accuracy and higher **Tok**. TreeRL exhibits pronounced regime sensitivity: in our evaluation, it provides little to no improvement on base models and mixed Think/Non-Think regimes. Moreover, even in the instruction-tuned Non-Think setting—commonly considered a favorable regime for TreeRL—it still falls short of TreeAdv, achieving lower accuracy while consuming more generated tokens (**Tok**). Overall, TreeAdv delivers more consistent gains across model regimes, improving both accuracy and token efficiency under comparable supervision and decoding settings.

4.3 Training Dynamics and Stability

We next analyze training-time dynamics to understand how the improvements emerge. Figure 4 compares TreeAdv with GRPO/GSPO across model scales, reporting reward (top) and entropy (bottom) over training steps; entropy serves as a proxy for policy sharpness and training-time volatility.

On `Qwen3-8B-Base`, TreeAdv achieves faster

Method	AIME24 (32p1)	AIME25 (32p1)	BRUMO (32p1)	CMI (32p1)	GPQA	HMMT (32p1)	MATH	OlymE	OlymH	Omni	Avg ↑ (%)	Tok ↓ (#)
`Qwen3-8B-Base`	11.25	9.48	18.85	4.45	18.69	1.35	68.20	7.00	3.00	24.07	16.64	2667
`w. DAPO`	31.56	25.94	38.85	16.48	41.41	14.79	87.60	18.00	4.00	35.50	31.41	6733
`w. TreeRL`	14.17	12.92	24.17	6.02	28.28	2.40	78.60	6.00	6.00	27.85	20.64	1945
`w. TreeAdv-GRPO`	27.71	24.79	37.19	16.09	49.49	11.77	85.60	20.00	10.00	34.03	31.67	6171
`Qwen3-8B-Inst Non-Think`	28.02	21.15	30.94	14.69	44.44	10.52	83.20	23.00	4.00	35.41	29.54	3506
`w. DAPO`	56.25	40.00	52.92	30.31	59.60	26.98	90.80	44.00	11.00	50.52	46.24	13373
`w. TreeRL`	28.33	21.67	30.42	15.63	47.47	11.88	83.80	23.00	2.00	34.62	29.88	3605
`w. TreeAdv-GRPO`	60.63	49.17	55.00	33.44	54.55	29.90	92.20	58.00	12.00	48.83	49.37	9962
`Qwen3-4B-Inst-2507`	61.56	45.73	**57.81**	32.03	60.10	30.31	93.40	57.00	11.00	49.55	49.85	7701
`w. DAPO`	61.56	51.04	57.29	31.72	66.16	**31.67**	93.00	58.00	10.00	50.61	51.11	6730
`w. TreeRL`	61.67	46.25	54.69	31.64	59.09	30.52	92.80	**59.00**	17.00	48.60	50.13	7540
`w. TreeAdv-GRPO`	**61.98**	**52.81**	56.98	**32.34**	**64.14**	**31.67**	**93.80**	53.00	**18.00**	**50.72**	**51.54**	5805

Table 2: Results on a suite of reasoning benchmarks comparing additional baselines, including DAPO and TreeRL, against the corresponding TreeAdv-GRPO-trained models. Tok is the average number of generated tokens per solution (lower is better). Green indicates better results compared with DAPO/TreeRL, while Red indicates worse results. For accuracy metrics, **Bold** indicates the highest value in each column.

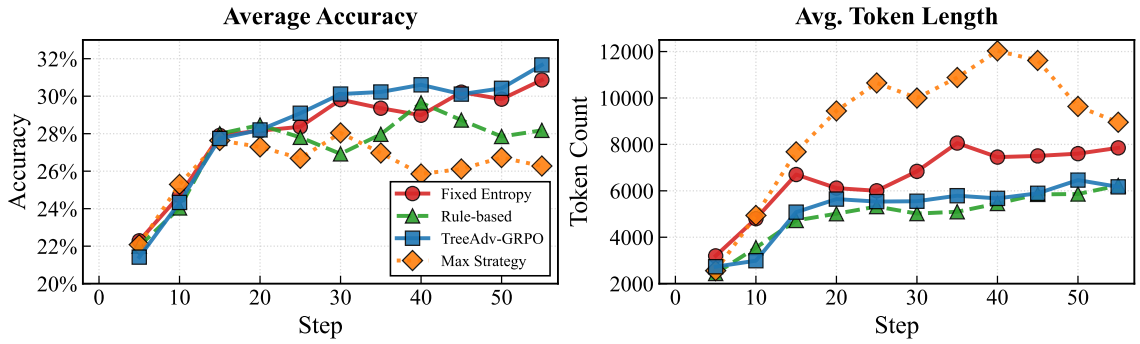

Figure 5: Evolution of average accuracy and token length across training steps under four different tree-branching strategies. We compare: Entropy Fixed (fixed entropy threshold H=1.4), Rule-based (branching at `.\n\n` and `.\n` instead of an entropy criterion), TreeAdv-GRPO (ours; default entropy-scheduled branching), and Max Strategy (using max instead of mean when aggregating descendant-leaf advantages). The x-axis denotes the training step; the left y-axis reports the mean accuracy averaged over 10 evaluation benchmarks, while the right y-axis reports the average generated token length.

early reward gains and a quicker entropy drop, after which the reward gap narrows later in training, consistent with earlier policy concentration under the same budget. On instruction-tuned models, TreeAdv often shows a short delay in early reward growth but consistently catches up and surpasses GRPO/GSPO, with typically smoother entropy trajectories, suggesting lower-variance updates from token-/segment-level advantage redistribution.

We further conduct checkpoint analyses on `Qwen3-8B-Base`. Figure 9 reports Pass@1 accuracy over training steps across all benchmarks, and Figure 3 zooms in on AIME 2025, BRUMO, and GPQA using dense checkpoints saved every 5 steps; accuracy broadly tracks reward, where TreeAdv improves steadily while GRPO shows a mid-training regression. Finally, Figure 8 jointly plots Pass@1 accuracy (solid) and average output length (dashed); between steps 20 and 40, TreeAdv improves accuracy without increasing output length, indicating gains beyond simply generating longer solutions.

5 Ablation

We ablate three components of TreeAdv under identical training data, rollout budget, decoding, and optimization settings (Figure 5). Specifically: (i) **Rule-based branching**, which replaces entropy-guided branching with delimiter-based splits at tokens such as `.\n\n` and `.\n`; (ii) **Max Strategy**, which replaces the mean in Eq. 3 with a max operator when aggregating descendant leaf advantages; and (iii) **Entropy Fixed**, which fixes the branching threshold at H=1.4 throughout training instead of using our schedule.

Figure 5 yields three takeaways. First, *where*

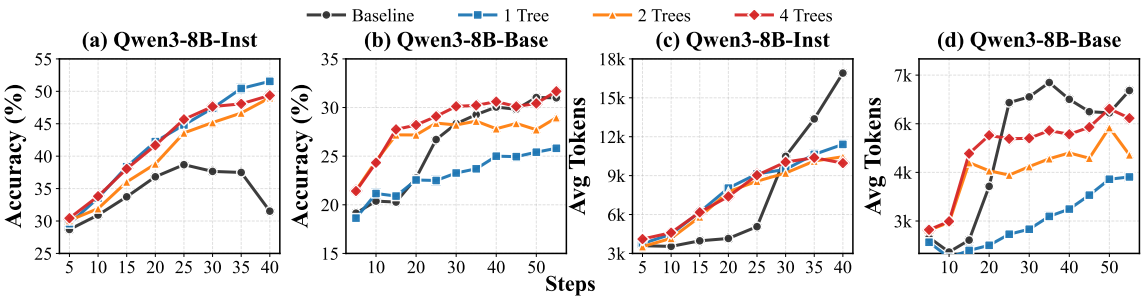

Figure 6: Performance and cost analysis across training steps. (a) and (b) illustrates the Average Accuracy trends, while (c) and (d) shows the corresponding Average Token consumption. The left column depicts the Qwen3-8B-Inst Non-think model, and the right column depicts the Qwen3-8B-Base model. Curves represent comparisons between the Baseline and varying tree search strategies (1 Tree, 2 Trees, and 4 Trees)

we branch matters: rule-based branching shows limited late-stage accuracy gains, suggesting that formatting boundaries are weaker proxies for high-uncertainty decision points than entropy-guided splits. Second, *how* we aggregate leaf advantages matters: Max Strategy improves faster early on but plateaus, indicating that emphasizing the single best descendant can produce a less stable training signal than averaging. Third, the entropy schedule has a stronger impact on efficiency than on accuracy: compared with Entropy Fixed, our adaptive threshold yields lower token usage in later training while maintaining comparable accuracy.

6 Analysis

6.1 Allocating a fixed rollout budget: sharing vs. exploration

Figure 6 analyzes training dynamics under a fixed rollout budget of $K{=}16$ completed trajectories per question. GRPO/GSPO sample K independent rollouts without shared prefixes. Our tree-structured variants allocate the same budget into $M \in \{1, 2, 4\}$ entropy-guided trees: smaller M increases prefix sharing, while larger M increases root-level diversity.

The optimal allocation is regime-dependent. On the instruct model (Figure 6a), $M{=}1$ yields the best and most stable accuracy gains, with $M{=}2$ and $M{=}4$ slightly behind. On the base model (Figure 6b), $M{=}4$ achieves the best final accuracy, while $M{=}1$ plateaus early. Overall, base models benefit more from root-level exploration, whereas instruct models benefit more from lower-variance learning signals enabled by stronger prefix sharing.

6.2 Accuracy versus token usage and stability

Token-usage trends indicate that gains are not simply driven by longer generations. On the instruct setting, GRPO becomes increasingly verbose at later steps while accuracy decreases (Figures 6a,c), whereas tree-structured variants keep token usage more controlled and are less prone to late-stage regression. On the base setting, token usage increases with larger M (Figure 6d), reflecting the cost of broader exploration; notably, the best-performing configuration ($M{=}4$) is also the most token-expensive. Appendix A.5 (Figure 7) reports rollout token statistics across additional model scales.

7 Conclusion

This work studies how to improve RL fine-tuning under a fixed rollout budget by changing how rollouts are sampled and how advantage is assigned. We propose TreeAdv, including tree-structured sampling which organizes K completed trajectories into M entropy-guided trees, and a structure-aware token-level advantage redistribution that uses outcome differences among continuations sharing the same prefix. This yields more comparable learning signals on shared segments and reduces spurious reinforcement of non-informative tokens.

Across model families, our method consistently increases average accuracy while *reducing* average token usage under the same rollout budget, demonstrating that gains are not driven by longer generations. Training is also more stable across epochs and settings, with lower variance and fewer late-stage regressions.

Limitations

Our experiments primarily focus on math-heavy long-form reasoning benchmarks and a small set of Qwen-family backbones. While TreeAdv is not inherently math-specific, additional evaluations on other domains (e.g., code, tool use, dialogue) and model families are needed to characterize generality.

TreeAdv also depends on practical rollout-tree construction choices (entropy thresholding, branching constraints, and a no-branch token list). These heuristics may require tuning across decoding settings, and our entropy estimate is approximated from top-k probabilities.

Finally, TreeAdv introduces extra systems complexity (tree/forest rollout management and segment-level aggregation). Although it often reduces redundant generation, realizing wall-clock gains may require careful implementation and scaling studies.

References

Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. 2025. Matharena: Evaluating llms on uncontaminated math competitions.

Brown University Math Olympiad. 2025. Brown university math olympiad (BrUMO) 2025. https://www.brumo.org. Accessed: 2025-12-23.

Lang Cao, Renhong Chen, Yingtian Zou, Chao Peng, Huacong Xu, Yuxian Wang, Wu Ning, Qian Chen, Mofan Peng, Zijie Chen, Peishuo Su, and Yitong Li. 2026. More bang for the buck: Process reward modeling with entropy-driven uncertainty. *Preprint*, arXiv:2503.22233.

Carnegie Mellon Informatics and Mathematics Competition. 2025. Carnegie mellon informatics and mathematics competition (CMIMC) 2025. https://cmimc.math.cmu.edu. Accessed: 2025-12-23.

Guanzhong Chen, Shaoxiong Yang, Chao Li, Wei Liu, Jian Luan, and Zenglin Xu. 2025. Heterogeneous group-based reinforcement learning for llm-based multi-agent systems. *Preprint*, arXiv:2506.02718.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 181 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Ruomeng Ding, Chaoyun Zhang, Lu Wang, Yong Xu, Minghua Ma, Wei Zhang, Si Qin, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. 2024. Everything of thoughts: Defying the law of penrose triangle for thought generation.

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, Zhengyang Tang, Benyou Wang, Daoguang Zan, Shanghaoran Quan, Ge Zhang, Lei Sha, Yichang Zhang, Xuancheng Ren, Tianyu Liu, and Baobao Chang. 2025. Omni-math: A universal olympiad level mathematic benchmark for large language models.

Google DeepMind. 2025. Gemini 3 pro: Model card. Model card update: December 2025; model release: November 2025.

Zhiwei He, Tian Liang, Jiahao Xu, Qiuzhi Liu, Xingyu Chen, Yue Wang, Linfeng Song, Dian Yu, Zhenwen Liang, Wenxuan Wang, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. 2025. Deepmath-103k: A large-scale, challenging, decontaminated, and verifiable mathematical dataset for advancing reasoning. *Preprint*, arXiv:2504.11456.

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.

Zhenyu Hou, Ziniu Hu, Yujiang Li, Rui Lu, Jie Tang, and Yuxiao Dong. 2025. Treerl: Llm reinforcement learning with on-policy tree search. *Preprint*, arXiv:2506.11902.

Lingjie Jiang, Xun Wu, Shaohan Huang, Qingxiu Dong, Zewen Chi, Li Dong, Xingxing Zhang, Tengchao Lv, Lei Cui, and Furu Wei. 2025. Think only when you need with large hybrid-reasoning models. *Preprint*, arXiv:2505.14631.

Jiazheng Li, Yawei Wang, David Yan, Yijun Tian, Zhichao Xu, Huan Song, Panpan Xu, and Lin Lee Cheong. 2025a. Salt: Step-level advantage assignment for long-horizon agents via trajectory graph. *Preprint*, arXiv:2510.20022.

Yizhi Li, Qingshui Gu, Zhoufutu Wen, Ziniu Li, Tianshun Xing, Shuyue Guo, Tianyu Zheng, Xin Zhou, Xingwei Qu, Wangchunshu Zhou, Zheng Zhang, Wei Shen, Qian Liu, Chenghua Lin, Jian Yang, Ge Zhang, and Wenhao Huang. 2025b. Treepo: Bridging the gap of policy optimization and efficacy and inference efficiency with heuristic tree-based modeling. *CoRR*, abs/2508.17445.

Huan Ma, Jingdong Chen, Joey Tianyi Zhou, Guangyu Wang, and Changqing Zhang. 2025. Estimating llm uncertainty with evidence. *Preprint*, arXiv:2502.00290.

Mathematical Association of America. 2024. American invitational mathematics examination (AIME) 2024. https://www.maa.org/math-competitions/aime. Accessed: 2025-12-23.

Mathematical Association of America. 2025. American invitational mathematics examination (AIME) 2025. https://www.maa.org/math-competitions/aime. Accessed: 2025-12-23.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel J. Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, pages 20275–20321. Association for Computational Linguistics.

Datta Nimmaturi, Vaishnavi Bhargava, Rajat Ghosh, Johnu George, and Debojyoti Dutta. 2025. Predictive scaling laws for efficient grpo training of large reasoning models. *Preprint*, arXiv:2507.18014.

OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, and 244 others. 2024. Openai o1 system card. *Preprint*, arXiv:2412.16720.

OpenAI. 2024. Introducing openai o1-preview.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, Michael Choi, Anish Agrawal, Arnav Chopra, Adam Khoja, Ryan Kim, Richard Ren, Jason Hausenloy, Oliver Zhang, Mantas Mazeika, and 1093 others. 2025. Humanity's last exam. *Preprint*, arXiv:2501.14249.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2023. Gpqa: A graduate-level google-proof q&a benchmark. *Preprint*, arXiv:2311.12022.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. 2016. High-dimensional continuous control using generalized advantage estimation.

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

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

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2025. Hybridflow: A flexible and efficient RLHF framework. In *Proceedings of the Twentieth European Conference on Computer Systems, EuroSys 2025, Rotterdam, The Netherlands, 30 March 2025 - 3 April 2025*, pages 1279–1297. ACM.

Haoxiang Sun, Yingqian Min, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, Zheng Liu, Zhongyuan Wang, and Ji-Rong Wen. 2025. Challenging the boundaries of reasoning: An olympiad-level math benchmark for large language models. *Preprint*, arXiv:2503.21380.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2024. Math-shepherd: Verify and reinforce llms step-by-step without human annotations.

Xuezhi Wang and Denny Zhou. 2024. Chain-of-thought reasoning without prompting. *CoRR*, abs/2402.10200.

Yutong Wang, Pengliang Ji, Chaoqun Yang, Kaixin Li, Ming Hu, Jiaoyang Li, and Guillaume Sartoretti. 2025. Mcts-judge: Test-time scaling in llm-as-a-judge for code correctness evaluation. *Preprint*, arXiv:2502.12468.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025a. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Zhicheng Yang, Zhijiang Guo, Yinya Huang, Xiaodan Liang, Yiwei Wang, and Jing Tang. 2025b. Treerpo: Tree relative policy optimization. *Preprint*, arXiv:2506.05183.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models.

Chenlu Ye, Zhou Yu, Ziji Zhang, Hao Chen, Narayanan Sadagopan, Jing Huang, Tong Zhang, and Anurag Beniwal. 2025. Beyond correctness: Harmonizing process and outcome rewards through rl training. *Preprint*, arXiv:2509.03403.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, and 16 others. 2025. Dapo: An open-source llm reinforcement learning system at scale. *Preprint*, arXiv:2503.14476.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. 2024. Rest-mcts*: LLM self-training via process reward guided tree search.

Kaiyan Zhang, Yuxin Zuo, Bingxiang He, Youbang Sun, Runze Liu, Che Jiang, Yuchen Fan, Kai Tian, Guoli Jia, Pengfei Li, Yu Fu, Xingtai Lv, Yuchen Zhang, Sihang Zeng, Shang Qu, Haozhan Li, Shijie Wang,

Yuru Wang, Xinwei Long, and 20 others. 2025. A survey of reinforcement learning for large reasoning models. *Preprint*, arXiv:2509.08827.

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

A Appendix

A.1 Entropy Details

No-branch token list. In practice, some high-entropy positions correspond primarily to formatting or structural tokens (e.g., whitespace, LaTeX control symbols, or bracket/brace delimiters). Branching at such positions rarely yields semantically meaningful alternatives and can unnecessarily increase the branching factor. We therefore maintain a *no-branch token list*: even when the entropy criterion $H(s_t) > \tau$ is satisfied, we suppress branching if the token at position t matches any entry in this list (see the tcolorbox below).

\, $, \n, \r, ␣, _, ␣␣, :, \(, \), \[, \], \{, \}, (,), [,], {, }

Entropy estimation with truncated log-probabilities. We obtain rollouts and token log-probabilities via vLLM, which returns log-probabilities only for the top-k candidates (with $k = 20$ in our setup). As a result, we approximate the token entropy using this truncated support rather than the full vocabulary:

$$\tilde{H}(s_t) \;=\; -\sum_{a \in \mathcal{A}_{20}(s_t)} \pi_{\text{old}}(a \mid s_{<t}) \log \pi_{\text{old}}(a \mid s_{<t}), \tag{8}$$

where $\mathcal{A}_{20}(s_t)$ denotes the set of top-20 tokens returned by vLLM at state s_t (tokens outside $\mathcal{A}_{20}(s_t)$ are treated as having zero probability under this approximation). We use $\tilde{H}(s_t)$ for entropy-guided branching in all experiments.

A.2 Training Details

Training data. All RL experiments are trained on the same supervision pool to enable a controlled comparison. We sample a fixed 10K subset from the DeepMath103K corpus and reuse it for all GRPO, GSPO, and TreeAdv runs. The subset is *not* reshuffled across methods: each method observes the same examples in the same order. This shared data schedule simplifies the analysis of optimization dynamics and helps isolate the effect of the learning algorithm.

Optimization and learning-rate schedule. All methods are implemented in the VERL framework, with optimization settings matched across objectives whenever possible. For all `Base` models, we use a peak learning rate of 5×10^{-6}; for all `Inst`

models, we use 1×10^{-7}. The minimum learning rate is set to 1×10^{-7} in all runs. Training lasts 57 optimization steps with a global batch size of 512, corresponding to approximately 3 epochs over the 10K subset. We use 2 warmup steps with a linear ramp from 0 to the peak learning rate, followed by cosine decay to the minimum learning rate. This schedule (total steps, warmup length, and cosine decay) is kept identical for GRPO, GSPO, TreeAdv-GSPO and TreeAdv-GRPO.

Checkpointing and model selection. We save checkpoints every 5 optimization steps throughout training. For each method and model, we evaluate every saved checkpoint and report the best-performing checkpoint on the held-out benchmarks in the main results. This early-stopping-style selection is applied uniformly to GRPO, GSPO, TreeAdv-GSPO and TreeAdv-GRPO, ensuring that each method is compared at its best point along an otherwise shared training trajectory.

TreeAdv-specific hyperparameters. TreeAdv introduces a small set of additional hyperparameters beyond the GRPO/GSPO baselines. We use an initial entropy threshold of $H_0 = 1.4$ to determine branching points in the tree. The threshold is annealed with a decrement of 0.05 over training, with a floor at $H_{\min} = 1.0$. Unless otherwise specified, for each prompt TreeAdv constructs 4 trees. Each prompt is allotted a total of 16 rollouts, distributed across these trees, and this tree/rollout configuration is held fixed across all TreeAdv experiments. To avoid premature branching at the very beginning of generation, we additionally impose an earliest-branching constraint: branching is only allowed at the first position that satisfies the entropy threshold *after* encountering one of the following delimiters in the generated text: `.\n\n`, `,` , or `.\n`.

Our entropy-guided branching is inspired by prior work on uncertainty-guided exploration (Cao et al., 2026). That work further reports an empirical analysis of token-level entropy distributions across datasets and difficulty strata (see its appendix A.12), which motivates our use of an annealed entropy threshold and the earliest-branching constraint.

Sampling for rollouts and inference. During training rollouts, we use top-k and nucleus (top-p) sampling with $k = 20$ and $p = 0.7$. We use the same sampling configuration for test-time inference on the evaluation benchmarks (top-k = 20, top-p = 0.7) to keep generation stochasticity consistent between training and evaluation.

Reward design and other settings. Reward design and other RL-specific settings (e.g., correctness scoring, shaping, and penalty terms) follow the recommended configuration in the DeepMath103K release. We use the same reward function for GRPO, GSPO, TreeAdv-GSPO and TreeAdv-GRPO so that observed differences are attributable to the training objective and advantage construction rather than changes in supervision. To prevent excessively long generations, we apply a length penalty consistently across all experiments: if the output exceeds 16K tokens, we set the original reward to -1.

DAPO Training For fair comparison, we follow DAPO's recommended training recipe and keep all optimization hyperparameters unchanged; the only difference is that we use a $2\times$ larger rollout budget to instantiate the tree-structured rollouts (all remaining settings strictly match those in DAPO).

TreeRL For fair comparison, we adopt TreeRL's default recommended configuration $(M, N, L, T) = (6, 2, 1, 2)$; aside from this, we keep the training pipeline and all remaining hyperparameters exactly the same as in the above setup.

Computational resources. The comprehensive experimental suite, including all training phases, hyperparameter tuning, and inference evaluations, was executed on a high-performance cluster equipped with **128 NVIDIA A800 GPUs**. The total computational cost for this study, covering all baselines and our proposed method, amounted to approximately **1,500 GPU hours**.

A.3 Test datasets details

We evaluate our models on a diverse collection of ten benchmarks, ranging from standard high school mathematics to graduate-level scientific reasoning and recent Olympiad competitions. The details of each dataset are as follows:

Standard Mathematics

- **MATH500**: A widely adopted subset of the MATH dataset, consisting of 500 randomly sampled problems. It covers seven subject areas (e.g., algebra, number theory, calculus)

and serves as a baseline for measuring standard mathematical problem-solving capabilities.

Olympiad-Level Competitions This category includes both broad-coverage benchmarks and specific high-difficulty contests from the 2024–2025 season to prevent data contamination and test frontier reasoning.

- **OmniMath**: A comprehensive Olympiad-level benchmark designed to assess advanced mathematical reasoning. We utilize the full test set consisting of 4,428 problems, which aggregates diverse competitions to ensure broad coverage of Olympiad-style challenges.
- **AIME 2024 & 2025**: The American Invitational Mathematics Examination (AIME) is a high-prestige intermediate Olympiad. We include all problems from both the AIME I and AIME II exams for each year, resulting in 30 problems for AIME 2024 and 30 problems for AIME 2025. These questions require integer answers between 000 and 999.
- **HMMT 2025**: The Harvard-MIT Mathematics Tournament (February 2025). We evaluate on a subset of 30 problems, specifically comprising the Algebra & Number Theory, Geometry, and Combinatorics subject rounds (10 problems each). This dataset represents one of the most challenging undergraduate-organized competitions for high school students.
- **BRUMO 2025**: The Brown University Math Olympiad (2025). We evaluate on a total of 30 problems, combining the complete Individual Round (15 problems) and the Team Round (15 problems). This dataset serves as a fresh evaluation benchmark with diverse problems across algebra, combinatorics, geometry, and number theory.
- **CMINMC 2025**: The Carnegie Mellon Informatics and Mathematics Competition (2025). We evaluate on a total of 40 problems, comprising the three Individual Rounds (Algebra & Number Theory, Combinatorics & Computer Science, and Geometry) and the Team Round, with 10 problems each. This dataset covers a hybrid of mathematical and algorithmic reasoning tasks.
- **OlymMATH**: A benchmark designed to challenge the reasoning boundaries of LLMs. We evaluate on two sampled subsets to assess performance across difficulty gradients:
  - **OlymMATH-Easy**: A subset of **100** problems representing entry-level Olympiad difficulty.
  - **OlymMATH-Hard**: A subset of **100** high-complexity problems, selected to test the models' capability in handling intricate logical deductions and multi-step reasoning.

Scientific Reasoning

- **GPQA-Diamond**: A subset of the Graduate-Level Google-Proof Q&A benchmark (Rein et al., 2023). It contains 198 multiple-choice questions written by domain experts in biology, physics, and chemistry. This dataset is designed to be difficult even for human experts to answer without access to external resources, serving as a proxy for high-level scientific reasoning.

A.4 Test Datasets Token Statistics

This subsection provides detailed token-usage statistics corresponding to Table 1. Table 3 reports, for each benchmark, the average number of generated tokens per solution (#; lower is better), which serves as a fine-grained proxy for generation-time compute and verbosity. We organize results into three groups: **(A)** representative SOTA open-weight models to calibrate typical token budgets; **(B)** controlled comparisons on base models; and **(C)** controlled comparisons on instruct models. Within each controlled comparison group, bold numbers indicate the more token-efficient method (i.e., fewer generated tokens). These per-benchmark statistics complement the aggregate **Tok** reported in the main table and help verify that improvements in the accuracy–efficiency frontier reflect broadly reduced redundant generation rather than being driven by a small subset of tasks.

A.5 Rollout Token Statistics

Figure 7 shows the training-time evolution of the *average number of tokens per question* measured over the rollouts collected during training. On Qwen3-4B-Inst-2507, the GRPO baseline exhibits a clear trend toward higher token usage as training progresses, while TreeAdv keeps the per-question

Model / Method	AIME24 (#)	AIME25 (#)	BRUMO (#)	CMI (#)	GPQA (#)	HMMT (#)	MATH (#)	OlymE (#)	OlymH (#)	Omni (#)	Avg (#)
(A) SOTA open models (Qwen & DeepSeek)											
Qwen3-32B-Inst Think	13137	15680	13993	18622	8086	17736	4485	15424	20951	12800	14091
Qwen2.5-7B-Inst	1999	1344	1251	1190	724	1151	661	1096	1252	1239	1191
Qwen2.5-Math-7B-Inst	1464	1290	1243	1284	1102	1216	662	1401	1260	1042	1196
Qwen2.5-QwQ-32B	13902	15970	13799	18656	8872	17481	4190	15941	20613	12331	14176
DeepSeek-R1-Distill-Qwen-7B	12629	13648	11558	15544	6922	15275	3370	15188	17378	10708	12222
DeepSeek-R1-Distill-Qwen-14B	11190	13551	11242	15774	6965	15552	3812	14653	17568	10873	12118
DeepSeek-R1-Distill-Qwen-32B	11339	13351	10265	14892	6459	14968	3715	12929	17524	10586	11603
DeepSeek-R1-Distill-Llama-8B	13772	14188	12767	16124	7727	15472	3570	15713	18025	11176	12854
(B) Our TreeAdvon base models											
Qwen3-8B-Base	4336	2948	3264	2753	2287	3054	1163	1912	2416	2540	2667
w. GRPO	9466	8919	6909	8600	1917	8807	2004	9182	8533	5911	7025
w. TreeAdv-GRPO	8256	7218	6460	7057	3909	7385	2047	7321	6818	5244	6171
w. GSPO	13527	11894	10158	11924	1951	13309	2202	11377	11868	7615	9583
w. TreeAdv-GSPO	12194	9695	8732	9000	3818	11381	2345	12064	7872	6086	8319
Qwen3-4B-Base	4343	3217	2481	2567	2096	3348	1377	3003	1067	2780	2628
w. GRPO	5963	3141	3792	2531	1760	3386	1001	2015	1997	2360	2794
w. TreeAdv-GRPO	8222	6941	6357	6187	1653	7920	2083	7398	6962	5220	5894
w. GSPO	12259	8626	6509	6932	3138	7693	1742	7535	4425	4597	6346
w. TreeAdv-GSPO	14467	10840	9129	8947	3222	11976	2906	10055	8232	6285	8606
(C) Our TreeAdv on instruct models											
Qwen3-4B-Inst Think	14441	17233	15469	20303	9211	18488	5029	17475	21994	13578	15322
w. GRPO	14382	17331	15009	19634	9184	18190	5020	16400	21603	13186	15000
w. TreeAdv-GRPO	11927	12930	15474	13243	17687	8008	16230	4552	15313	19030	13439
Qwen3-4B-Inst Non-Think	5381	4040	3170	4163	2622	3476	1108	3462	3002	2899	3333
w. GRPO	12612	12473	7313	13816	4959	12984	1958	11139	11616	8225	9710
w. TreeAdv-GRPO	11072	9878	6168	11215	4575	10783	1685	8787	9533	6459	8015
Qwen3-8B-Inst Think	15001	18017	16378	20775	9862	20066	5472	18667	23282	14481	16200
w. GRPO	15039	17524	16066	20363	9450	19604	5317	17035	22212	14319	15693
w. TreeAdv-GRPO	11825	13736	12112	15503	6978	14977	4093	13615	17324	10570	12073
Qwen3-8B-Inst Non-Think	5973	4078	3369	4149	2530	4190	1224	3553	2912	3081	3506
w. GRPO	13540	13816	12089	17871	12874	16933	3162	15457	17402	10585	13373
w. TreeAdv-GRPO	12361	12991	10416	15377	5580	15064	2132	13766	17114	9291	11409
w. GSPO	11021	11076	8209	13229	4303	11629	2044	11864	13011	7814	9420
w. TreeAdv-GSPO	12893	13378	10657	15356	5469	14962	2316	15373	17553	9543	11750
Qwen3-4B-Inst-2507	8814	8379	7095	9820	5374	9555	1726	9893	10051	6304	7701
w. GRPO	11093	11055	8882	14011	6481	15091	2023	12199	15286	7852	10397
w. TreeAdv-GRPO	6072	6346	5483	6847	4621	6810	1806	7423	7371	5268	5805
w. GSPO	7129	7014	5989	7665	4435	8188	1605	7925	7975	5396	6332
w. TreeAdv-GSPO	6265	6960	5820	7470	4978	7262	1886	7725	7936	5670	6197
Qwen3-30B-A3B-2507	6114	6895	6031	7666	5142	7638	1610	6890	9212	5713	6291
w. GSPO	5712	6185	5502	6807	4314	6840	1414	6499	7930	5125	5633
w. TreeAdv-GSPO	5641	6390	5601	7039	4403	6879	1450	6388	8532	5156	5748

Table 3: Token efficiency on olympiad-level reasoning benchmarks. We report the average number of generated tokens per solution (lower is better). Green indicates the more efficient method (lower token count) within each controlled comparison group.

token count comparatively stable. On Qwen3-8B-Base and Qwen3-30B-Inst-2507, TreeAdv follows token-usage dynamics similar to the corresponding baselines (GRPO and GSPO, respectively). Overall, the results indicate that TreeAdv does not rely on systematically longer generations during training, and its behavior is consistent across model scales. Since the training-time compute cost is approximately proportional to the number of processed tokens (ignoring first-token latency and other constant overheads), these token-usage trends also suggest comparable training efficiency between TreeAdv and the baselines.

A.6 Training Accuracy and Output Length on Qwen3-8B-Base

We analyze the training dynamics of TreeAdv (ours) and GRPO (baseline) on Qwen3-8B-Base from two complementary views. Figure 8 presents an aggregate comparison: solid curves show Pass@1 accuracy over training steps (left axis), while dashed curves report the average generated output length in tokens (right axis). TreeAdv achieves higher accuracy than GRPO throughout

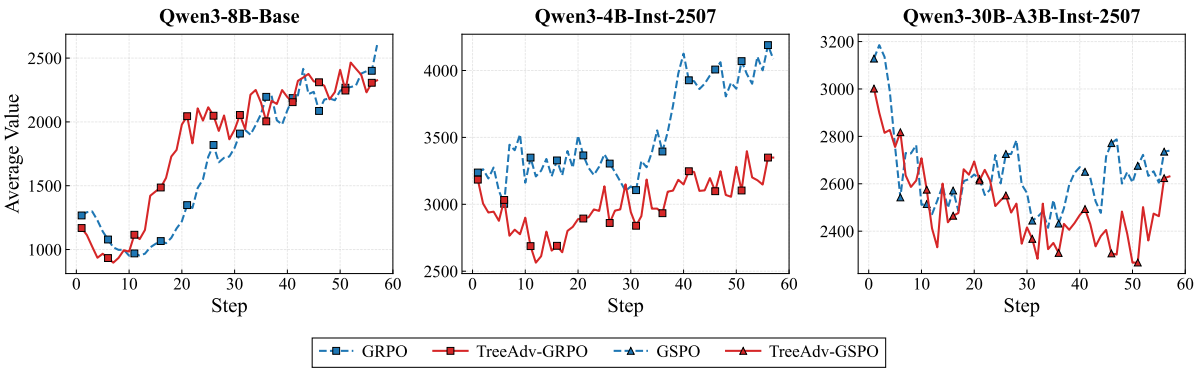

Figure 7: Comparative analysis of TreeAdv versus GRPO and GSPO across model scales. The curves report the *average number of tokens per question during training rollouts* (averaged over the fixed rollout budget per question) for Qwen3-4B-Inst-2507, Qwen3-8B-Base, and Qwen3-30B-Inst-2507. TreeAdv (red) yields token usage that is comparable to or lower than the baselines (blue), indicating that its gains are not driven by longer rollouts.

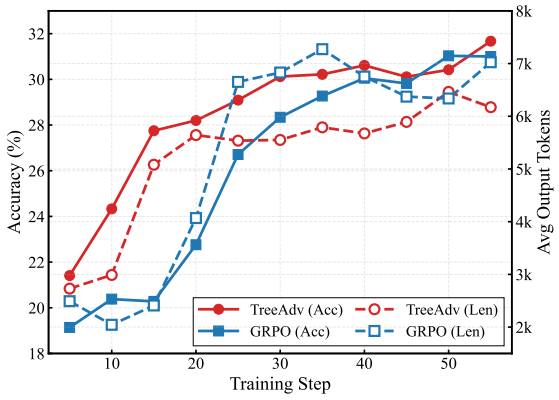

Figure 8: Training dynamics comparison between TreeAdv (Ours) and GRPO (Baseline). The solid lines represent the accuracy (left axis), while the dashed lines indicate the average output token length (right axis). TreeAdv (red) consistently outperforms the baseline (blue) in accuracy while maintaining comparable reasoning lengths.

training and improves more steadily. Meanwhile, the output-length curves remain close, suggesting that the gains are not primarily explained by systematically longer generations, but by more effective optimization under comparable output lengths.

To verify that this advantage is not driven by a single dataset, Figure 9 further reports Pass@1 accuracy trajectories across ten mathematical reasoning benchmarks. Across diverse datasets (e.g., AIME 2024/2025, OmniMath, and GPQA Diamond), TreeAdv-GRPO consistently converges faster and reaches a higher final accuracy than the GRPO baseline.

A.7 Training Dynamics and Inference Latency Analysis

In this section, we provide a detailed analysis of the training stability of `Qwen3-8B-Instruct` in the Non-thinking Mode. To ensure the reliability of our observations, we conducted three independent runs for the baseline GRPO. Figure 10 illustrates the dual-axis comparison between these baseline runs and our proposed **TreeAdv-GRPO**, tracking both the training reward (solid lines, left axis) and the average wait count (dashed lines, right axis).

Consistent Collapse of Baseline GRPO. A critical and reproducible observation is the severe instability inherent in the baseline GRPO. As shown in Figure 10, all three baseline runs exhibit a catastrophic collapse in the later stages of training. We observe a strong negative correlation where the degradation in training reward coincides perfectly with a sudden explosion in inference latency.

Specifically, the average wait count for the baseline models spikes drastically, reaching values exceeding 100 (peaking at ≈ 137 in one run). This significantly surpasses the visualization limit of the y-axis (truncated at 20 for clarity). This phenomenon suggests that without proper regularization, the baseline model falls into degenerate behaviors—generating excessively long, repetitive, or meaningless reasoning chains ("over-thinking" without logic)—which consumes substantial computational resources while actively harming answer correctness.

Stability of TreeAdv-GRPO. In sharp contrast, **TreeAdv-GRPO** (red) demonstrates superior robustness under the exact same experimental set-

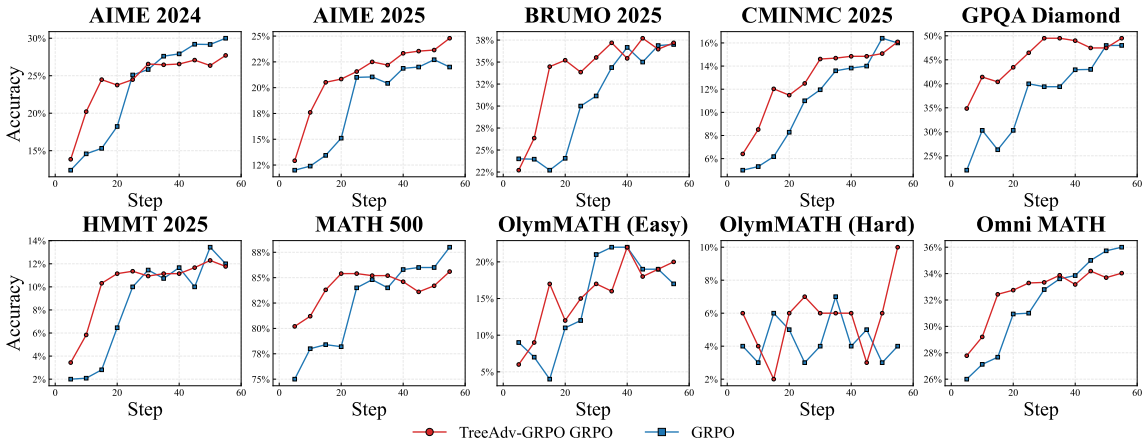

Figure 9: Training dynamics on ten mathematical reasoning benchmarks. We compare the performance of our proposed TreeAdv-GRPO (red circles) against the GRPO baseline (blue squares). The curves illustrate the Pass@1 accuracy evolution over training steps. Across diverse datasets—including challenging competitions like AIME 2024/2025 and comprehensive benchmarks like Omni MATH and GPQA Diamond—TreeAdv-GRPO consistently demonstrates superior convergence speed and higher final accuracy compared to the baseline.

tings. It effectively suppresses the tendency for latency explosion, maintaining the average wait count consistently below 10 throughout the entire training process. This indicates that our method successfully regularizes the model's reasoning path, preventing the emergence of inefficient long-context patterns and ensuring steady reward growth without the risk of mode collapse observed in the baselines.

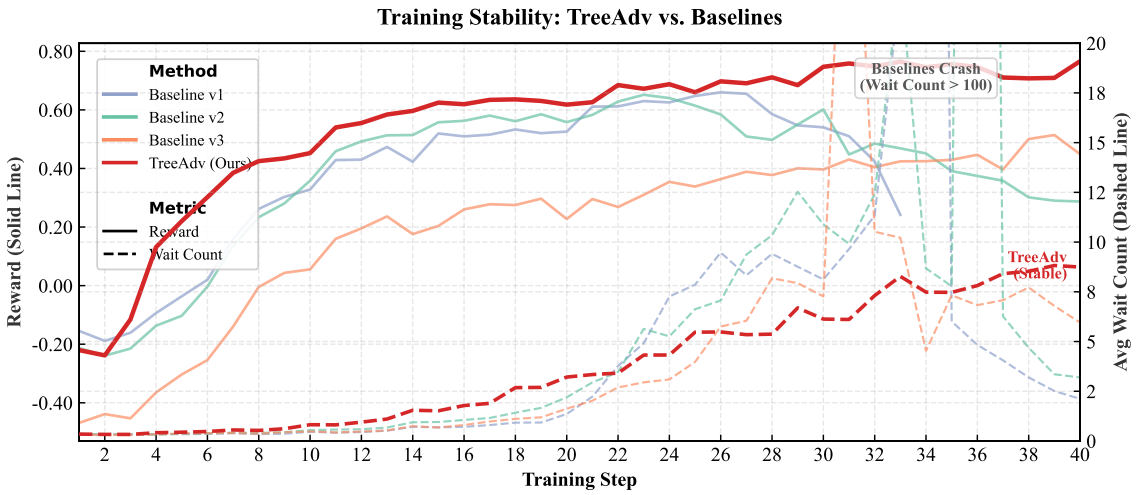

Figure 10: Comparison of training stability on Qwen3-8B-Instruct (Non-thinking mode). Solid lines represent the reward (left axis), while dashed lines denote the average wait count (right axis). All three baseline runs eventually suffer from a "wait count explosion" (exceeding 100, truncated here for visibility), leading to model collapse. In contrast, TreeAdv maintains robust stability with a consistently low wait count throughout the training process.