IS TREE RL LOSING TO STANDARD RL? VALIDATION MISMATCH IN TREE-STRUCTURED RL

Anonymous authors
Paper under double-blind review

ABSTRACT

Tree-structured reinforcement learning methods branch rollouts from shared prefixes, reusing generated tokens and turning outcome rewards into process supervision, while reporting consistent gains over GRPO. Against DAPO, a more recent post-training RL algorithm that corrects the entropy collapse and instability of GRPO, these gains are no longer observed under standard validation: across three tree-RL methods, three model scales and four tasks, with the number of training trajectories per update matched, the tree-trained policies trail the DAPO-trained policies in all five comparisons, by 1.0 to 4.6 points. We identify a *validation mismatch* in this assessment between how training rollouts are generated and how checkpoints are validated. Standard validation evaluates every checkpoint using greedy decoding or independent samples, the procedure used in standard RL training, but not in tree-based RL training. Tree validation, which evaluates checkpoints with the branching procedure that tree RL trains with, corrects the assessment. Under tree validation, the tree-trained policy outperforms the DAPO-trained policy in all five comparisons, by 0.9 to 4.3 points, revealing training progress that standard validation misses. On Qwen2.5-Math-1.5B-Instruct, TreeRL shows no improvement over its initialization under standard validation but improves by 4.5 points under tree validation. The same pattern holds at deployment: in every setting we test, a tree-trained policy deployed with a tree has the highest mean accuracy of the six combinations of training method and inference procedure, while generating fewer tokens than a majority vote over independent samples. Tree RL should therefore be validated and monitored with the procedure it is trained with, alongside standard validation, with the generator, answer selector and budget of each procedure specified.

1 INTRODUCTION

Reinforcement learning with verifiable rewards (RLVR) has become a standard stage in post-training language models for reasoning. The most common RL algorithm, GRPO (Shao et al., 2024), samples a group of independent responses for each prompt, scores each response as a whole and normalizes rewards within the group. Tree-structured RL changes how these rollouts are generated. Instead of sampling responses independently, methods such as TreeRL, TreePO and Tree-GRPO branch new continuations from prefixes of earlier responses (Hou et al., 2025; Li et al., 2025; Ji et al., 2026). Branching reuses the tokens already generated for a shared prefix, and comparing the outcomes of sibling branches turns a single terminal reward into credit for intermediate steps. These methods report consistent improvements over GRPO.

These improvements, like those of any RL recipe, are measured through validation. During RL training, practitioners monitor a validation curve by periodically evaluating the current checkpoint on a held-out benchmark. The resulting accuracy is used to compare training recipes and track training progress. The checkpoint is evaluated with a fixed decoding strategy, typically greedy decoding, or k independent samples scored by their average accuracy or their majority vote. We call this *standard validation*. The tree-RL methods we survey follow this practice without exception (Table 1): TreeRL, Tree-GRPO and SPO-tree (Guo et al., 2025) report greedy accuracy, TreePO a majority vote over sixteen samples, and TreeRPO (Yang et al., 2025) a mean over sixteen samples. Although these methods generate their training rollouts through branching, they are validated solely

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

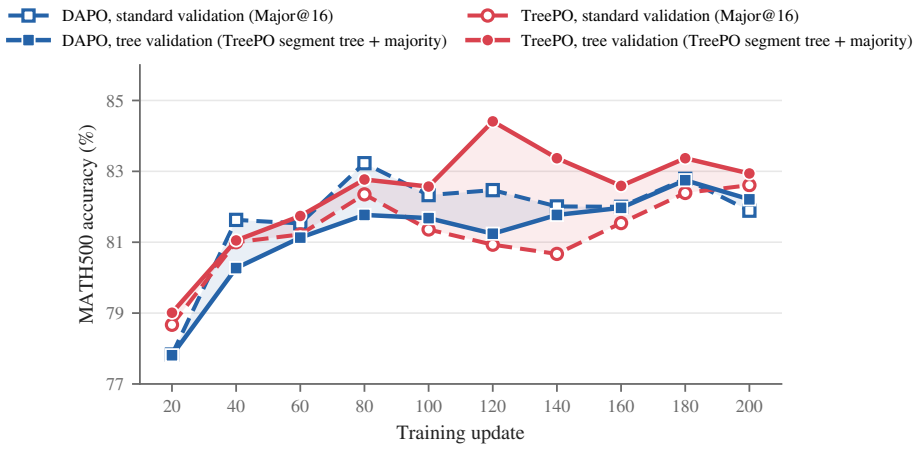

Figure 1: **Tree training progresses, but only aligned validation shows it.** TreePO vs DAPO on MATH500 with Qwen2.5-7B, mean over three seeds (standard deviations in Table 4). Red: tree rollout (TreePO); blue: standard rollout (DAPO). Dashed lines: standard validation, a majority vote over 16 independent samples, as reported by TreePO. Solid lines: tree validation, TreePO's segment-tree sampler followed by the same vote, which matches how TreePO generates its training rollouts. Under standard validation, TreePO appears no better than DAPO and declines by 1.7 points between updates 80 and 140; under tree validation, the same checkpoints show no such decline, reach their highest accuracy at update 120, and stay above DAPO at every checkpoint. Shading marks the gap between the two validation procedures for each policy.

Method	Where the tree branches	Credit from the tree	Validation procedure
TreeRL	highest-entropy tokens	descendant success rate	greedy
TreePO	fixed-length segments	subgroup-calibrated	maj@16
Tree-GRPO	agent steps	intra- and inter-tree groups	greedy
TreeRPO	fixed-length steps	mean over children	mean@16, T=0.6
SPO-tree	every 30 tokens	advantage vs. siblings	greedy, short context

Table 1: **Five tree-structured RL methods and their validation procedures.** All train by branching and validate without branching, with greedy decoding or independent samples. Transcribed from each paper's configuration and main results.

without branching, using either greedy decoding or independent sampling; GRPO also generates its rollouts without branching.

Tree RL's reported gains are established against GRPO, yet standard RL has since advanced. DAPO, for example, identifies entropy collapse, reward noise and training instability in its initial GRPO baseline and addresses these issues through changes to clipping, sampling and loss design, raising AIME 2024 accuracy on Qwen2.5-32B from 30 to 50 points (Yu et al., 2025). Whether tree RL retains its advantage over this stronger baseline remains an open question. We therefore compare three tree-RL methods with DAPO across three model scales and four tasks, using shared initializations and matching the number of training trajectories per update. Under standard validation, tree RL's advantage disappears: tree-trained policies underperform their DAPO-trained counterparts in all five comparisons, by up to 4.6 points (Table 2; Figure 1).

These results do not contradict earlier reports. Prior tree-RL methods were compared against GRPO, whose training suffers from entropy collapse and instability, as reported by DAPO, and whose accuracy can degrade later in training, as reported in the TreeRL study. Improvements over a degrading baseline may therefore obscure a subtler, systematic effect: a *validation mismatch* (Figure 2). Standard validation generates candidates without branching, aligning with DAPO's training rollout procedure but differing from that of tree RL. The comparison thus evaluates DAPO under a procedure consistent with its training and tree RL under a different one. This mismatch also matters at deployment: when served with standard decoding, tree-trained policies operate under the procedure

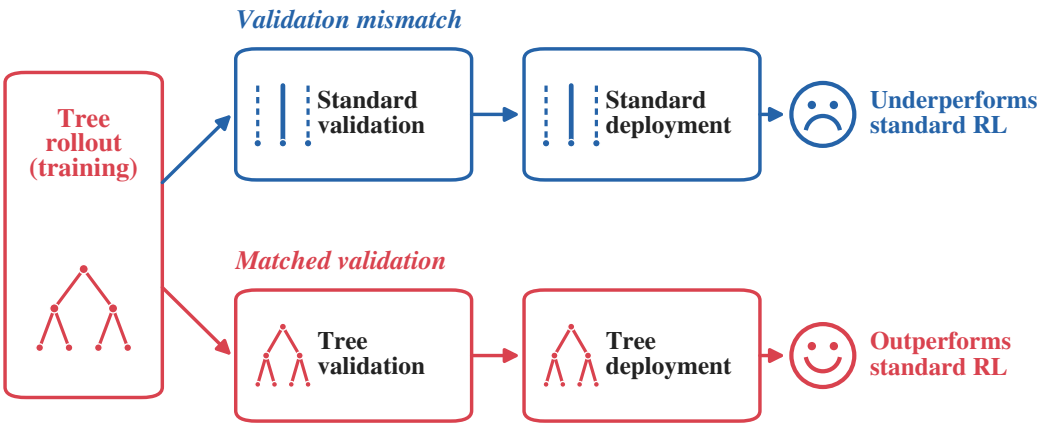

Figure 2: **Validation mismatch in tree-structured RL.** Validated and deployed without branching, the tree-trained policy underperforms standard RL (top); validated and deployed with the branching used in training, it outperforms standard RL under the same procedure (bottom).

in which they underperform DAPO. To measure its effect, we additionally validate every checkpoint with *tree validation*, which generates the same number of candidates with the branching procedure used in tree training and scores them in the same way as standard validation. Applying both evaluations to both policies yields a 2×2 design that crosses *tree versus standard training* with *tree versus standard validation*, in which the DAPO-trained policy receives exactly the same tree procedure as its tree-trained counterpart.

Aligned validation changes the assessment of tree RL. Under tree validation, the tree-trained policy outperforms the DAPO-trained policy in all five comparisons, by 0.9 to 4.3 points (Figure 3). Because both procedures draw 16 samples and score them in the same way, the two differ only in the generator, and each generator favors the policy trained with it. Aligned validation also tracks training progress that standard validation misses. In the most pronounced case, TreeRL on Qwen2.5-Math-1.5B-Instruct, the standard-validation accuracy of the tree-trained policy ends 0.2 points below its initialization, so standard validation reports no progress, while its accuracy under tree validation rises by 4.5 points.

These results carry over to deployment (Section 4.5). With the number of samples, the budget and the selector held fixed, replacing independent samples with a tree raises the accuracy of the tree-trained policy by 2.3 to 5.1 points and changes that of DAPO by -2.3 to $+0.4$ points. In every deployment setting, tree training with tree deployment has the highest mean accuracy, and it generates 15% to 29% fewer tokens than DAPO with independent samples.

This paper is an empirical analysis of how tree RL is validated. We make three contributions:

- We show that the reported gains of tree RL over GRPO are no longer observed against DAPO under standard validation, across three tree-RL methods, three model scales and four tasks.
- We identify a validation mismatch in how tree RL is evaluated: standard validation evaluates tree-trained policies without the branching procedure they are trained with, and thereby understates both their performance and their training progress; tree validation, aligned with training, corrects both, and results on tree RL must therefore state the validation procedure.
- We show that at deployment, tree training with tree deployment has the highest mean accuracy of any combination of training method and inference procedure in every setting we test, while generating fewer tokens than standard RL with independent samples, and propose validating tree RL with the procedure it is trained with.

The rest of the paper is structured as follows. Section 2 describes tree-structured RL, situates our work with respect to prior work and defines the components of an evaluation procedure. Section 3

describes the matched training pairs and the validation and deployment procedures. Section 4 reports the comparison under both validation procedures, the measured training progress, the deployment results and the obstacles to serving tree inference, and Section 5 outlines three practices for validating tree RL.

2 BACKGROUND

2.1 TREE-STRUCTURED RL

The methods in Table 1 differ in detail but share a common structure. For a question x, the policy first produces one or more root responses. The procedure then selects positions inside already-generated text at which to fork, and samples new continuations from each forked prefix. Methods differ in where the fork is placed — at the highest-entropy tokens (TreeRL), at fixed-length segment boundaries (TreePO, SPO-tree), at the end of a complete agent step (Tree-GRPO) — and in how many rounds of forking they run. The result is a bank of K terminal paths per question whose prefixes overlap.

Credit is propagated through the shared prefixes. A terminal path receives a binary outcome reward from a verifier. An internal node is then valued by its descendants: the fraction of its descendant leaves that are correct, or the mean value of its children (TreeRL, TreeRPO). A segment's advantage is that value compared against its parent, its root or its siblings, and the resulting scalar is broadcast to every token of the segment before a clipped policy-ratio update. Section 3 describes the training runs.

2.2 RELATED WORK

Training for the inference procedure. Inference-aware methods tailor the training objective to a specific inference procedure. InfAlign shows that standard RLHF is suboptimal under best-of-n and improves inference-time win rate by calibrating the reward to that procedure (Balashankar et al., 2025), while related works optimize best-of-n or test-time compute directly (Chow et al., 2025; Ousherovitch & Tewari, 2026). Related training–inference mismatches arise in exposure bias (Ross et al., 2011; Bengio et al., 2015) and pass@k optimization degrading pass@1 (Barakat et al., 2026). AlphaGo Zero is trained and evaluated with tree search, and its policy network alone is substantially weaker than the full system (Silver et al., 2017). Pass@k analyses likewise show that conclusions about RL can depend on the sampling budget (Yue et al., 2025).

Harness and environment. For LLM agents, the harness plays the role of the inference procedure: post-training under one tool environment or scaffold can lose much of its gain under another (Kim et al., 2026; Thangarajah et al., 2026), and recent work trains with the harness in the loop (Chen et al., 2026; Yu et al., 2026; He et al., 2026; Ning et al., 2026).

Tree-structured RL. Tree-structured RL descends from process supervision over intermediate steps (Lightman et al., 2024), while VinePPO estimates the value of intermediate states with Monte Carlo rollouts from them (Kazemnejad et al., 2025). TreePO reports test-time scaling with its tree sampler, and MARS^2 notes that rankings change with the inference configuration (Li et al., 2026). Neither, however, compares tree-trained and standard-RL policies under identical tree-based and independent-sampling procedures, as we do here.

2.3 WHAT AN EVALUATION PROCEDURE SPECIFIES

An evaluation procedure is used for validation during training and for deployment after training. It has three parts: a *generator* that produces a bank of candidates with their decoding distributions; a *budget*; and an *endpoint* that turns the bank into the quantity scored. We call the procedure that decodes without branching, by greedy decoding or independent sampling, *standard validation* or *standard deployment*, and the procedure that generates with the branching procedure used in training (Algorithm 1) *tree validation* or *tree deployment*. Throughout, *tree rollout* and *standard rollout* refer to how training rollouts are generated, by tree RL and by DAPO, respectively, and *tree validation* and *standard validation* refer to how checkpoints are evaluated. Tree validation reuses the complete

rollout procedure of the tree method, including its fork rule and its segment, depth and termination rules, with the tree size set to the validation budget (Table 6). We call an evaluation *aligned* with a training method when its generator is the one that method trains with: standard validation for DAPO, tree validation for a tree-RL method. The generator alone does not fix the reported number. Training uses every terminal path and selects none, whereas an evaluation that returns one answer must specify a selector, and the average correctness of candidates is not a substitute for the accuracy of the returned answer: a selector that outperforms a single sample can rank two policies differently from a ranking based on their candidate means (Appendix B). We therefore state the selector of every procedure, and distinguish the *trajectory-level mean*, the average correctness over all paths of a tree, from the *trajectory-level majority vote*, the accuracy of the answer returned by a majority vote over those paths.

3 Experimental design

Training pairs. Every comparison pairs a tree-RL method with DAPO, trained from the same initialization on the same data, prompts and update schedule. We use DAPO as the baseline because it outperforms GRPO with the same base model and mitigates entropy collapse, which can limit the branch diversity on which tree methods depend. We evaluate five paired comparisons: TreeRL on Qwen2.5-Math-1.5B-Instruct (MATH500) and on Qwen2.5-3B (LiveCodeBench), Tree-GRPO on Qwen2.5-3B (HotpotQA), and TreePO on Qwen2.5-7B (AIME 2024–25 and MATH500). Each tree-RL method uses its official configuration, and both runs within each pair use the same number of training trajectories per update. As in the tree-RL literature, each pair compares two complete training recipes. The TreePO MATH500 pair is trained for 200 updates and the others for 100 (Appendix A).

Validation. Every checkpoint of both policies is evaluated using two procedures: standard validation, which draws 16 independent samples at the training temperature (IID16); and tree validation, which grows a tree with 16 terminal paths using the fork rule of the pair's tree method (Algorithm 1). Both are scored by the mean accuracy over the 16 samples (mean@16; for a tree, the trajectory-level mean) or by the accuracy of their majority vote (maj@16). Appendix A lists the scoring used for each pair. Each point is the mean over three training seeds, and error bars show ± 1 standard deviation across seeds.

Deployment. We deploy the TreeRL policies on Qwen2.5-Math-1.5B-Instruct at update 100, and the TreePO policies of the MATH500 pair at updates 100 and 200 on MATH-4.5K, the 4,500 problems of the MATH test split (Hendrycks et al., 2021) that are not in MATH500; this split is disjoint from the training and validation data. Each procedure returns one answer per question: greedy decoding, or a majority vote over six independent samples (IID6) or over a six-path tree (Tree6), at temperature 0.7 with an 8K-token budget (4K context length for Qwen2.5-Math-1.5B-Instruct).

4 Results

4.1 Under standard validation, tree RL trails DAPO

Table 2 collects the five comparisons. Under standard validation, the tree-trained policy performs worse than the DAPO-trained policy in every comparison. The same holds under maj@16 on LiveCodeBench and HotpotQA (Figure 5).

4.2 Under tree validation, tree RL leads DAPO

Tree validation evaluates each tree-trained policy with the branching procedure it was trained with, and evaluates the DAPO-trained policy with the same procedure (Table 2 and Figure 3; full curves in Figures 4 and 5).

Under tree validation, the tree-trained policy leads DAPO in all five comparisons. This advantage also holds under alternative scoring rules (mean@16 on the 1.5B pair, maj@16 on AIME; Figure 4). On the TreePO MATH500 pair, it leads at all ten checkpoints to update 200, by 1.1 points on average, against -0.5 points under standard validation (Figure 1).

Algorithm 1 Tree validation and tree deployment. FORK is the fork rule of the method: highest-entropy tokens (TreeRL), segment boundaries (TreePO) or agent steps (Tree-GRPO). $y_{<j} \circ y'$ is the prefix of y before token j followed by a new continuation y', and ANSWER extracts the final answer of a path. The trajectory-level mean averages the correctness of the answers in $\mathcal{A}$ instead of applying S. Shared prefixes are generated once.

Require: policy π_θ; question x
Require: M roots; L rounds; N fork points per tree; T continuations per fork point
Require: fork rule FORK; selector S (majority vote)
Ensure: one returned answer $\hat{a}$

for $m = 1, \dots, M$ **do** ▷ sample the roots
  $\mathcal{T}_m \leftarrow \{y_m\}$, where $y_m \sim \pi_\theta(\cdot \mid x)$
end for
for $\ell = 1, \dots, L$ **do** ▷ grow each tree
  for $m = 1, \dots, M$ **do**
    for each fork point $(y, j) \in \text{FORK}(\mathcal{T}_m, N)$ **do**
      sample T continuations $y' \sim \pi_\theta(\cdot \mid x, y_{<j})$
      add each path $y_{<j} \circ y'$ to $\mathcal{T}_m$
    end for
  end for
end for
$\mathcal{A} \leftarrow \{\text{ANSWER}(y) : y \in \mathcal{T}_1 \cup \cdots \cup \mathcal{T}_M\}$ ▷ $K = M(1 + LNT)$ answers
return $\hat{a} = S(\mathcal{A})$

	Standard validation		Tree validation		Δ final		Δ mean		
Setting	DAPO	Tree	DAPO	Tree	Std.	Tree	Std.	Tree	Shift
TreeRL, 1.5B, MATH500	73.4 ± 0.9	70.4 ± 0.5	74.2 ± 0.8	**77.9** ± 0.7	-3.0	$+3.7$	-1.4	$+2.5$	$+6.7$
TreeRL, 3B, LCB	**35.2** ± 2.4	30.6 ± 2.1	25.7 ± 2.4	30.0 ± 1.2	-4.6	$+4.3$	-2.1	$+1.6$	$+8.9$
Tree-GRPO, 3B, HotpotQA	20.9 ± 1.5	19.2 ± 1.0	19.2 ± 1.1	**21.5** ± 0.8	-1.7	$+2.3$	-1.2	$+1.0$	$+4.0$
TreePO, 7B, AIME	**22.0** ± 0.7	18.0 ± 1.0	15.0 ± 1.0	16.5 ± 1.7	-4.0	$+1.5$	-2.4	$+1.1$	$+5.5$
TreePO, 7B, MATH500	82.3 ± 0.7	81.4 ± 1.4	81.7 ± 0.5	**82.6** ± 1.2	-1.0	$+0.9$	-0.4	$+0.9$	$+1.9$

Table 2: **Tree-trained and DAPO-trained policies under standard and tree validation.** Validation results at update 100, mean $\pm$ standard deviation over three seeds; the highest accuracy in each row is in bold. Δ is tree-trained minus DAPO-trained accuracy (points) under standard (Std.) and tree validation, at update 100 (final) and averaged over updates 20–100 (mean). Shift is the final Δ under tree validation minus that under standard validation. Both procedures use 16 samples scored by mean@16, or by maj@16 for the MATH500 pairs of TreeRL and TreePO.

4.3 TREE VALIDATION TRACKS THE PROGRESS OF TREE TRAINING

Validation curves are also used to monitor training progress. From a shared initialization, standard validation measures a larger improvement for DAPO than for tree training in all four comparisons with an initial checkpoint, whereas tree validation measures a larger improvement for tree training (Table 3, Appendix C). The curves in Figures 4 and 5 show that the same ordering holds for most of the training rather than only at the end.

The difference is most visible on the 1.5B pair: standard validation reports that the TreeRL policy has not improved over its initialization, whereas tree validation shows a gain of 4.5 points on the same checkpoints (Figure 4a). When tree deployment is the target, standard validation understates both the progress of tree training and its standing relative to the standard-RL baseline; tree validation is the appropriate signal for monitoring. When standard deployment is the target, the standard validation curve is the relevant one, and it records a deficit.

4.4 EACH VALIDATION PROCEDURE FAVORS THE POLICY TRAINED WITH IT

Standard and tree validation draw the same number of samples and score them in the same way, so they differ only in how the samples are generated. A sampler effect common to both policies, such as that of its budget or termination rules, would leave their difference unchanged. Instead, in

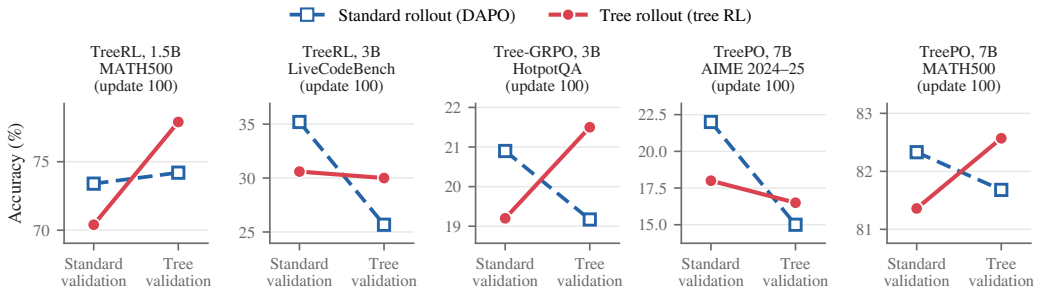

Figure 3: **Mismatched and aligned validation of all five comparisons.** Validation results at update 100 (Table 2). In every panel, DAPO is higher under standard validation and the tree-trained policy is higher under tree validation.

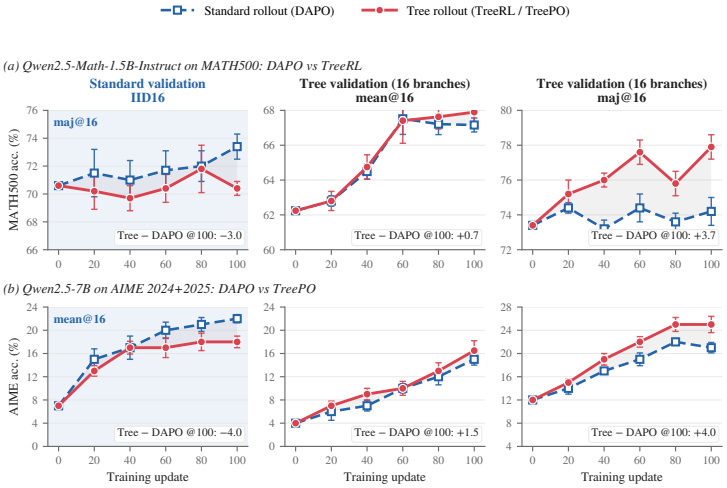

Figure 4: **Validation curves of two pairs.** Validation results, mean ± standard deviation over three seeds; insets give the tree-minus-DAPO gap at update 100. Left (shaded): standard validation, IID16 scored by maj@16 in (a) and mean@16 in (b). Middle and right: tree validation with 16 branches, scored by mean@16 and maj@16. **(a)** TreeRL vs DAPO, Qwen2.5-Math-1.5B-Instruct, MATH500. **(b)** TreePO vs DAPO, Qwen2.5-7B, AIME 2024–25.

every comparison (Table 2), switching from standard to tree validation shifts the tree-minus-DAPO difference in favor of the tree-trained policy, by 1.9 to 8.9 points and by 5.4 ± 0.7 points on average (standard error; Appendix A). The mismatch acts in both directions: tree validation evaluates the DAPO-trained policy with a generator it was not trained with, and its accuracy falls in four of the five comparisons. Each policy therefore performs best relative to the other under the procedure it was trained with, the pattern expected from a rollout–validation mismatch rather than from a sampler that is uniformly stronger or weaker. The effect appears under all three fork rules, entropy-based forking (TreeRL), segment-based forking (TreePO) and agent-step forking (Tree-GRPO), so it does not stem from a single sampler implementation. In the TreeRL MATH500 pair, tree validation improves the accuracy of both policies while favoring the tree-trained one.

The pattern is not confined to the final checkpoint. Averaged over all evaluated checkpoints, the tree-minus-DAPO difference is negative under standard validation and positive under tree validation in all five comparisons, and the tree-trained policy is above DAPO under tree validation at 22 of 25 checkpoints (Figures 1, 4 and 5).

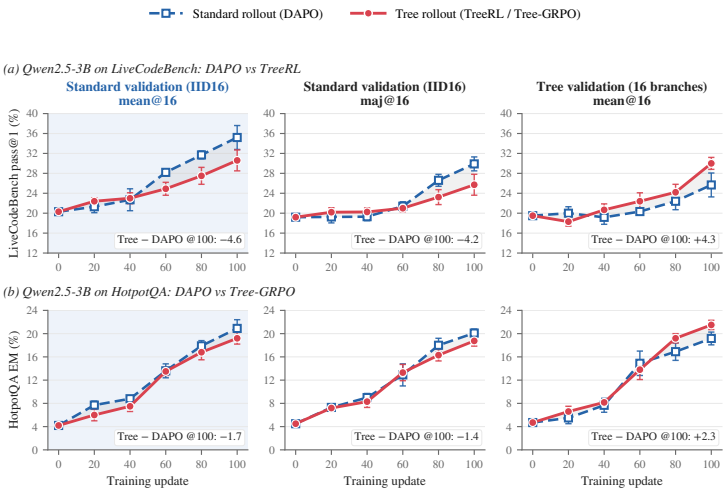

Figure 5: **Validation curves of two pairs on Qwen2.5-3B.** Validation results, mean ± standard deviation over three seeds. Left (shaded) and middle: standard validation, IID16 scored by mean@16 and maj@16. Right: tree validation with 16 branches, mean@16. **(a)** TreeRL vs DAPO, LiveCodeBench (pass@1). **(b)** Tree-GRPO vs DAPO, HotpotQA (exact match).

4.5 Tree training pays off at deployment

Validation shows which training method is better under each procedure; deployment asks which combination of training method and inference procedure serves the most accurate answer at an acceptable cost. We deploy the TreeRL and TreePO policies on MATH-4.5K, which is disjoint from both the training data and the validation sets, under three procedures that each return one answer (Figure 6).

Tree deployment favors the policy trained with the tree. With the number of samples, the budget, the temperature and the selector held fixed, replacing six independent samples with a six-path tree raises the accuracy of the tree-trained policy by 2.3 to 5.1 points and changes that of DAPO by -2.3 to $+0.4$ points. The benefit of branching therefore depends on how the policy was trained.

Standard deployment favors DAPO. With greedy decoding and with a majority vote over independent samples, DAPO leads the tree-trained policy in all three settings, as under standard validation.

Tree training with tree deployment has the highest mean accuracy, with fewer generated tokens. In every setting, the tree-trained policy deployed with a tree has the highest mean accuracy of the six configurations. It exceeds DAPO with independent samples by 0.7 to 1.8 points while generating 15% to 29% fewer tokens per question, because the tree paths share prefixes. For DAPO as well, tree deployment generates 19% to 28% fewer tokens than independent samples.

4.6 Tree deployment in current serving systems

Deployed systems typically return a single greedy or low-temperature response per request. In this regime, tree training offers no advantage: under greedy decoding, the tree-trained policy trails DAPO in all three deployment settings (Figure 6). Its advantage appears only under tree inference (Appendix D), for which current serving systems are not built. Inference engines such as vLLM and SGLang (Kwon et al., 2023; Zheng et al., 2024) schedule requests as independent sequences, whereas tree inference must fork a request at intermediate positions, extend the paths segment by segment within a budget, and aggregate them into one answer. In the released TreePO implementation, these steps run outside the engine, so a request passes through as many sequential rounds of generation as the tree has levels. As long as the procedure under which tree training wins is not the procedure that is served, tree RL is unlikely to become the standard post-training recipe; this requires serving systems that treat branching, budgeted expansion and answer selection as primitives.

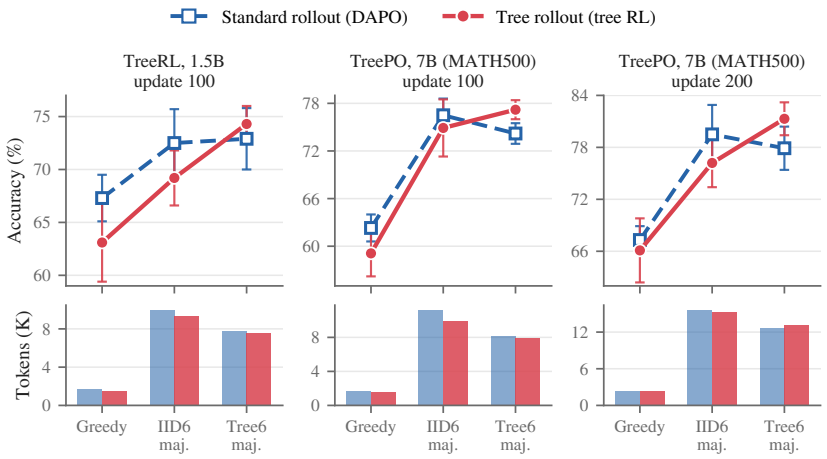

Figure 6: **Deployment on MATH-4.5K.** Deployment results, mean ± standard deviation over three seeds (values in Table 5). Top: accuracy under greedy decoding and majority votes over six independent samples (IID6) or six tree paths (Tree6). Bottom: newly generated tokens per question, in thousands.

4.7 Scope and limitations

Our study covers a limited set of tree-RL methods, model scales and tasks, and compares them against a single standard-RL recipe. Extending the comparison to other methods, larger models and additional domains would show how broadly the validation mismatch applies. A finer analysis of how individual components of the tree procedure contribute to the mismatch is left to future work.

5 Discussion: validating tree RL

Our results suggest three practices for evaluating tree RL. First, state the validation procedure, its generator, selector and budget, with every reported result, because the measured benefit of tree RL depends on it. Second, compare tree RL against a strong baseline such as DAPO, not only GRPO, and validate both policies with both standard and tree validation, with the same generator, selector and budget for both policies under each procedure; a comparison under one procedure evaluates one policy with its training generator and the other with a different generator. Third, monitor training with the procedure whose performance is the target. If tree deployment is the target, standard validation understates the progress of tree training, and tree validation should be tracked throughout. If standard deployment is the target, the deficit under standard validation is the relevant result, and tree training must reduce this deficit to be preferred.

The lesson is not specific to tree-based RL. We expect the same mismatch whenever the procedure that generates training rollouts departs from standard decoding, as with other structured or search-based rollout schemes: validating such a policy with standard decoding measures it under a procedure it was not trained with, and can misjudge both the training method and its progress. Assessing a training method therefore requires a validation procedure matched to its rollout procedure.

6 Conclusion

Under standard validation, tree RL appears to offer no benefit over DAPO, but this assessment reflects a validation mismatch between how tree RL generates its training rollouts and how its checkpoints are validated. Under tree validation, tree RL leads DAPO in all five comparisons, shows more training progress and has the highest mean accuracy at deployment. Tree RL should therefore be validated and monitored with the procedure it trains with, and every reported result should state the validation procedure that produced it.

AI USE STATEMENT

We used generative AI tools in two ways. First, to aid and polish the writing: drafting and editing parts of the text, improving clarity and grammar, and formatting the manuscript and references. Second, for retrieval and discovery: searching for and summarizing related work, which the authors verified before citing. The authors reviewed all AI-assisted content and take full responsibility for the content of this paper.

REFERENCES

Ananth Balashankar, Ziteng Sun, Jonathan Berant, Jacob Eisenstein, Michael Collins, Adrian Hutter, Jong Lee, Chirag Nagpal, Flavien Prost, Aradhana Sinha, Ananda Theertha Suresh, and Ahmad Beirami. InfAlign: Inference-aware language model alignment. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 2646–2672. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/balashankar25a.html.

Anas Barakat, Souradip Chakraborty, Khushbu Pahwa, and Amrit Singh Bedi. Why pass@k optimization can degrade pass@1: Prompt interference in LLM post-training, 2026. URL https://arxiv.org/abs/2602.21189.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, 2015.

Zhengyu Chen, Teng Xiao, Huaisheng Zhu, Yige Yuan, Luan Zhang, and Jingang Wang. Co-harness: Co-evolving harnesses and model weights for LLM agents, 2026. URL https://arxiv.org/abs/2607.22688.

Yinlam Chow, Guy Tennenholtz, Izzeddin Gur, Vincent Zhuang, Bo Dai, Aviral Kumar, Rishabh Agarwal, Sridhar Thiagarajan, Craig Boutilier, and Aleksandra Faust. Inference-aware fine-tuning for best-of-n sampling in large language models. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 78936–78959, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/c40bed606c51c8e827c1ba75aa2da054-Paper-Conference.pdf.

Yiran Guo, Lijie Xu, Jie Liu, Dan Ye, and Shuang Qiu. Segment policy optimization: Effective segment-level credit assignment in RL for large language models. In *Advances in Neural Information Processing Systems*, 2025. URL https://arxiv.org/abs/2505.23564.

Zhiyuan He, Siwei Zhang, Zhiwen Zhou, Yuqing Yang, Yu Kang, Yuge Zhang, Luna K. Qiu, Tin Yan Tsui, Jiahang Xu, and Chong Luo. Agent lightning v1.0: Towards harnessed agentic RL, 2026. URL https://arxiv.org/abs/2608.17528.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2021. URL https://arxiv.org/abs/2103.03874.

Zhenyu Hou, Ziniu Hu, Yujiang Li, Rui Lu, Jie Tang, and Yuxiao Dong. TreeRL: LLM reinforcement learning with on-policy tree search. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12355–12369, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.604. URL https://aclanthology.org/2025.acl-long.604/.

Yuxiang Ji, Ziyu Ma, Yong Wang, Guanhua Chen, Xiangxiang Chu, and Liaoni Wu. Tree search for LLM agent reinforcement learning. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2509.21240.

Amirhossein Kazemnejad, Milad Aghajohari, Eva Portelance, Alessandro Sordoni, Siva Reddy, Aaron Courville, and Nicolas Le Roux. VinePPO: Refining credit assignment in RL training of LLMs. In *International Conference on Machine Learning*, 2025. URL https://arxiv.org/abs/2410.01679.

Kyungmin Kim, Youngbin Choi, Seoyeon Lee, Suhyeon Jun, Dongwoo Kim, and Sangdon Park. The interplay of harness design and post-training in LLM agents, 2026. URL https://arxiv.org/abs/2606.25447.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023.

Pengfei Li, Shijie Wang, Fangyuan Li, Yikun Fu, Kaifeng Liu, Kaiyan Zhang, Dazhi Zhang, Yuqiang Li, Biqing Qi, and Bowen Zhou. MARS^2: Scaling multi-agent tree search via reinforcement learning for code generation. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 33319–33335, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1538. URL https://aclanthology.org/2026.acl-long.1538/.

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. TreePO: Bridging the gap of policy optimization and efficacy and inference efficiency with heuristic tree-based modeling, 2025. URL https://arxiv.org/abs/2508.17445.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2305.20050.

Xuying Ning, Dongqi Fu, Tianxin Wei, Hanqing Zeng, Yuanchen Bei, Bingxuan Li, Zihao Li, Qifan Wang, Xiang Shen, Yifan Wu, Jiayi Liu, Hong Li, Yinglong Xia, Xiangjun Fan, Hanghang Tong, and Jingrui He. EvoHarness-RL: Learning self-evolving runtime harness for long-horizon LLM agents, 2026. URL https://arxiv.org/abs/2608.05446.

Adam Ousherovitch and Ambuj Tewari. Compute aligned training: Optimizing for test time inference, 2026. URL https://arxiv.org/abs/2604.24957.

Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 627–635, 2011.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. HybridFlow: A flexible and efficient RLHF framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, 2025.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, 2017.

Kishanthan Thangarajah, Boyuan Chen, and Ahmed E. Hassan. DCAS: Decoupling CLI agent scaffolding to internalize planning across scaffolds, 2026. URL https://arxiv.org/abs/2608.06113.

Zhicheng Yang, Zhijiang Guo, Yinya Huang, Xiaodan Liang, Yiwei Wang, and Jing Tang. TreeRPO: Tree relative policy optimization, 2025. URL https://arxiv.org/abs/2506.05183.

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, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. DAPO: An open-source LLM reinforcement learning system at scale, 2025. URL https://arxiv.org/abs/2503.14476.

Xiao Yu, Baolin Peng, Ruize Xu, Hao Zou, Qianhui Wu, Hao Cheng, Wenlin Yao, Nikhil Singh, Zhou Yu, and Jianfeng Gao. OpenForgeRL: Train harness-native agents in any environment, 2026. URL https://arxiv.org/abs/2607.21557.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in LLMs beyond the base model? In *Advances in Neural Information Processing Systems*, 2025. URL https://arxiv.org/abs/2504.13837.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, 2024.

A Experimental details

Training. The DAPO runs use the DAPO recipe of verl (Sheng et al., 2025) with its default settings: decoupled clipping ($\epsilon_{\text{low}} = 0.2$, $\epsilon_{\text{high}} = 0.28$), a token-level policy-gradient loss, overlong reward shaping and dynamic sampling, without KL or entropy regularization. At every update, both runs of a pair train on the same number of prompts, with as many complete rollouts per prompt in the DAPO run as the tree-RL run has terminal paths, so the two runs update on the same number of trajectories; the tree-RL run generates fewer new tokens, since its paths share prefixes. The TreeRL pair on Qwen2.5-Math-1.5B-Instruct uses 1,024-token responses. Table 6 in Appendix D lists the tree configuration of each method.

Validation. The scoring of each pair is as follows.

- *TreeRL vs DAPO (1.5B, MATH500) and TreePO vs DAPO (7B, AIME).* Standard validation: IID16, scored by maj@16 for the 1.5B pair and by mean@16 for the AIME pair. Tree validation: scored by the trajectory-level mean and by the trajectory-level majority vote.
- *TreeRL vs DAPO (3B, LiveCodeBench) and Tree-GRPO vs DAPO (3B, HotpotQA).* Standard validation: IID16, scored by mean@16 and by maj@16. Tree validation: scored by mean@16.
- *TreePO vs DAPO (7B, MATH500).* Standard validation: the procedure reported by TreePO, a majority vote over IID16. Tree validation: TreePO's segment-tree sampler (Algorithm 2) followed by the same majority vote.

LiveCodeBench and HotpotQA are evaluated on their full evaluation sets. Majority votes are gold-free and cluster symbolically equivalent answers. Following the evaluation script of the TreePO repository, maj@k is estimated for each question as the majority-vote accuracy averaged over 1,000 bootstrap resamples, drawn with replacement, of the k samples; per-question scores, and hence accuracies, are therefore not restricted to multiples of $1/N$ for a benchmark of N questions. The same estimator is used for the majority votes of the deployment study. In Table 2, the MATH500 pairs are scored by maj@16, the validation metric TreePO reports on MATH500, and the other pairs by mean@16. Table 2 reports update 100, the last checkpoint shared by all five pairs.

Statistical uncertainty. For each comparison, the standard error of the Shift in Table 2 is computed from the seed standard deviations of its four entries, treating them as independent; this overstates the standard error when the two evaluations of the same checkpoints are positively correlated. The Shift is 6.7 ± 0.9, 8.9 ± 2.4, 4.0 ± 1.3, 5.5 ± 1.3 and 1.9 ± 1.2 points in the order of Table 2, and 5.4 ± 0.7 points on average over the five comparisons. It is positive in all five comparisons, at the final checkpoint and averaged over checkpoints.

B Candidate mean and returned-answer accuracy

Candidate mean and returned-answer accuracy need not order policies alike.

General statement. Write $p_i(x)$ for the probability that one sampled response of policy i is correct on question x, and $g(p)$ for the probability that the endpoint returns a correct answer when each candidate is correct with probability p, with $g(0) = 0$ and $g(1) = 1$; candidate mean (the trajectory-level mean when the candidates are the paths of a tree) is $\mathbb{E}_x[p_i(x)]$ and returned-answer accuracy is $\mathbb{E}_x[g(p_i(x))]$. For any such g with $g(t) \neq t$ at some t, the policies $p_1 = (1, 0)$ and $p_2 = (t, t)$ on two questions weighted $m = (t + g(t))/2$ and $1 - m$ satisfy

$$\big[\mathbb{E}[p_1] - \mathbb{E}[p_2]\big]\big[\mathbb{E}[g(p_1)] - \mathbb{E}[g(p_2)]\big] = -\tfrac{1}{4}\,(g(t) - t)^2 < 0, \tag{1}$$

so the two quantities agree on every pair of policies if and only if $g(p) = p$, a selector no better than a single sample, such as a uniformly random pick or always the first candidate. This is an existence result in the simplest setting, a scalar success rate per question and independent candidates; real tree selectors depend on how wrong answers cluster and on correlation between paths, and the construction does not indicate how often the orders differ. It shows that reporting tree-candidate averages in place of the returned answer does not remove the need to specify the selector.

Attribution. A uniform gain from extra inference compute, shared by both policies, is not evidence about tree training. Matched DAPO/tree pairs evaluated under the same procedures isolate the training effect, because both policies are evaluated with each procedure.

C Additional validation results

Table 3 gives the improvement of each policy from the shared initialization to update 100, for the four comparisons with an initial checkpoint (Section 4.3).

		Standard validation		Tree validation	
Pair	Model, task	DAPO	Tree	DAPO	Tree
TreeRL vs DAPO	1.5B, MATH500	+2.8	−0.2	+0.8	+4.5
TreeRL vs DAPO	3B, LCB	+14.9	+10.3	+6.2	+10.5
Tree-GRPO vs DAPO	3B, HotpotQA	+16.7	+15.0	+14.5	+16.8
TreePO vs DAPO	7B, AIME	+15.0	+11.0	+11.0	+12.5

Table 3: **Measured training progress depends on the validation procedure.** Validation results. Change in accuracy (points) from the shared initialization to update 100, under standard validation (IID16) and tree validation (16 branches) as in Table 2. Means over three seeds per method. The alternative tree-validation scoring agrees: TreePO gains 13 points of AIME maj@16 (DAPO: 9), and TreeRL 5.7 points of 1.5B mean@16 (DAPO: 4.9).

Table 4 reports every point of Figure 1 with its standard deviation across the three training seeds.

Table 5 lists the deployment results of Figure 6.

	Standard validation (maj@16)		Tree validation	
Update	DAPO	TreePO	DAPO	TreePO
20	77.83 ± 1.12	78.67 ± 0.75	77.81 ± 1.23	79.01 ± 0.75
40	81.63 ± 0.63	81.00 ± 0.97	80.27 ± 2.16	81.05 ± 1.27
60	81.53 ± 0.51	81.22 ± 1.27	81.13 ± 1.25	81.74 ± 1.45
80	83.23 ± 0.46	82.35 ± 1.14	81.77 ± 1.13	82.77 ± 0.76
100	82.33 ± 0.70	81.36 ± 1.38	81.68 ± 0.52	82.57 ± 1.21
120	82.47 ± 0.98	80.93 ± 2.10	81.24 ± 0.79	84.41 ± 0.93
140	82.01 ± 1.32	80.67 ± 1.67	81.77 ± 0.58	83.37 ± 0.59
160	82.00 ± 1.69	81.54 ± 2.15	81.97 ± 0.52	82.59 ± 1.14
180	82.79 ± 1.87	82.39 ± 1.24	82.75 ± 1.40	83.37 ± 0.57
200	81.89 ± 0.92	82.61 ± 1.23	82.21 ± 1.03	82.94 ± 0.84

Table 4: **Values of Figure 1.** Validation results on MATH500 with Qwen2.5-7B, accuracy (%) as mean ± standard deviation over three seeds.

		DAPO-trained		Tree-trained	
Setting	Deployment	Accuracy	Tokens	Accuracy	Tokens
TreeRL, 1.5B, MATH500, update 100	Greedy	67.3 ± 2.2	1.7K	63.1 ± 3.7	1.5K
	IID6 + majority	72.5 ± 3.2	10.0K	69.2 ± 2.6	9.3K
	Tree6 + majority	72.9 ± 2.9	7.8K	**74.3** ± 1.7	7.5K
TreePO, 7B, MATH500, update 100	Greedy	62.3 ± 1.7	1.7K	59.1 ± 2.9	1.6K
	IID6 + majority	76.5 ± 2.1	11.2K	74.9 ± 3.6	9.9K
	Tree6 + majority	74.2 ± 1.3	8.1K	**77.2** ± 1.2	7.9K
TreePO, 7B, MATH500, update 200	Greedy	67.3 ± 1.6	2.4K	66.1 ± 3.7	2.4K
	IID6 + majority	79.5 ± 3.4	15.6K	76.2 ± 2.8	15.2K
	Tree6 + majority	77.9 ± 2.5	12.6K	**81.3** ± 1.9	13.2K

Table 5: **Values of Figure 6.** Deployment results on MATH-4.5K, accuracy (%) as mean ± standard deviation over three seeds, and newly generated tokens per question. The highest mean accuracy in each setting is in bold.

D TREE CONFIGURATIONS AND TREE INFERENCE

Table 6 lists the tree configuration of each method in training, tree validation and tree deployment. Tree validation uses the fork rule of each method with 16 terminal paths, which matches IID16, and tree deployment uses six paths, which matches IID6.

Method	Fork rule	Training	Tree validation (16 paths)	Tree deployment (6 paths)
TreeRL	highest-entropy tokens	$M = 6$, $N = 2$, $T = 2$, $L = 1$ (30 paths)	$M = 4$, $N = 3$, $T = 1$, $L = 1$	$M = 2$, $N = 1$, $T = 2$, $L = 1$
TreePO	segment boundaries	$W = 16$, segment length 512 tokens, maximum depth 14, branching factor 2	same as training	$W = 6$, otherwise as training
Tree-GRPO	agent steps	official configuration (16 paths)	same as training	—

Table 6: **Tree configuration of each method.** M roots, N fork points per tree, T continuations per fork point and L rounds (Algorithm 1); W trajectories for TreePO (Algorithm 2).

Algorithm 2 summarizes the segment-tree sampler used for tree validation and tree deployment of the TreePO pairs. It is taken from the official TreePO repository: the rollout function `generate_sequences_tree_deepth_first_vanilla_mode` in `verl/workers/rollout/vllm_rollout/vllm_rollout_spmd.py`, with helpers in `recipe/treepo/vllm_rollout_tree.py`. TreePO uses the same function to generate its training rollouts, and the repository's offline generation entry point (`verl/trainer/main_generation.py` with `infer_mode=tree`) calls it for any checkpoint. With the tree parameters of the training configuration, the trees used for validation are therefore constructed in the same way as the trees used for training; only the final step differs, since training uses every trajectory whereas tree validation scores the trajectory-level mean or majority vote. Tree deployment uses the same sampler with six paths. In the released training loop, validation does not use this function: the tree sampler is disabled when a batch is marked for validation, and the released evaluation script samples sixteen independent responses.

TreeRL and Tree-GRPO also release their tree samplers: TreeRL's entropy-guided tree search (`openrlhf/trainer/ppo_utils/entropy_chain_local_manager.py`, parameters M, N, L, T) and Tree-GRPO's agent-step tree search (`search_r1/llm_agent/generation_ts.py`, `run_llm_loop_tree_search`). Both follow the structure of Algorithm 1 with their own fork rules, and in the released Tree-GRPO training loop validation likewise uses independent rollouts rather than the tree sampler.

Algorithm 2 Segment-tree inference of TreePO, as implemented in the official TreePO repository. Parameter values in parentheses are those of TreePO's released training configuration.

Require: policy π_θ; question x
Require: number of trajectories W (16); segment length G (512 tokens); maximum depth D (14); branching factor b (2)
Ensure: W trajectories $\mathcal{Y}$

$\mathcal{A} \leftarrow b$ copies of the empty response ▷ active paths
$\mathcal{Y} \leftarrow \emptyset$ ▷ finished paths
for $d = 1, \ldots, D$ **do**
  if $\mathcal{A} = \emptyset$ and $|\mathcal{Y}| < W$ **then** ▷ fallback
    fork new paths from random prefixes in $\mathcal{Y}$, preferring boxed answers with EOS
  end if
  extend every path in $\mathcal{A}$ by one segment of at most G tokens sampled from π_θ ▷ batched
  for each path $y \in \mathcal{A}$ **do**
    if y ends with EOS, repeats itself, reaches the length limit, or $d = D$ **then**
      move y from $\mathcal{A}$ to $\mathcal{Y}$
    end if
  end for
  $B \leftarrow \min\big(W - |\mathcal{Y}|,\ b\,(|\mathcal{A}| + n_d)\big)$ ▷ n_d: paths finished at depth d
  fork $\mathcal{A}$ into $\max(B, |\mathcal{A}|)$ paths, spread evenly ▷ branch at segment boundary
end for
return $\mathcal{Y}$