CAST: CREDIT ASSIGNMENT ON SELECTIVE REASONING TREES FOR LONG-CHAIN REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning with verifiable rewards has become the standard recipe for improving long-chain reasoning. However, current methods struggle to put compute and training signal where they matter most in a trajectory. On hard problems, an early mistake often derails the trajectory, and later reflection rarely fixes it. On easier problems, once the model has enough information to determine the correct answer, it keeps generating redundant verification and self-checks. Consequently, training signals get diluted across uninformative steps, and rollout compute is wasted on reasoning that no longer changes the outcome. To resolve both bottlenecks, we propose **C**redit **A**ssignment on **S**elective Reasoning **T**rees (CAST), a tree-structured RL framework for long-chain reasoning. During rollout, CAST builds selective reasoning trees, expanding only at unresolved branch points where sampled continuations yield mixed outcomes (both correct and incorrect). It then pools all leaf rewards and the tree topology to estimate intermediate step values in closed form, deriving fine-grained edge advantages. Across five diverse backbones, ranging from 1.7B base models to a 35B mixture-of-experts, CAST consistently improves accuracy while generating shorter responses. On Qwen3-4B-Base, CAST outperforms DAPO by 4.03 points on average, cuts response length by 25.8%, and matches AIME24 accuracy with a $1.8\times$ training speedup. By focusing rollouts on decisive steps and sharing credit across the tree, CAST shows that targeted exploration and structured credit assignment make long-chain reasoning both stronger and more efficient. Upon acceptance, we will release our code and model checkpoints.

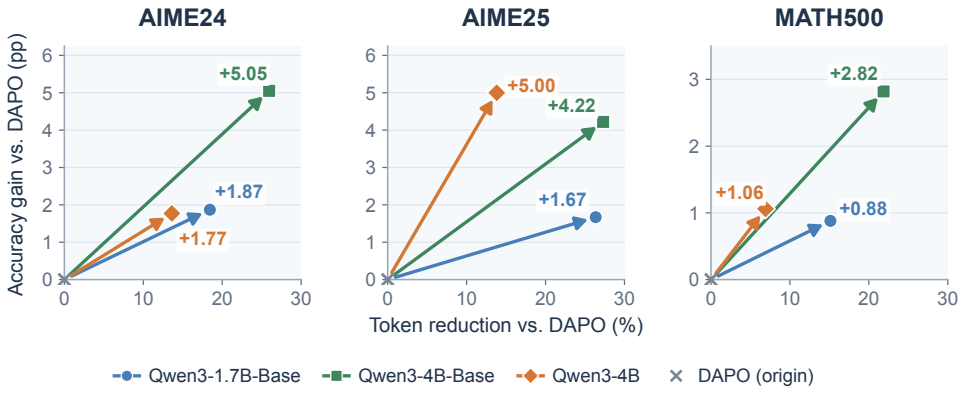

Figure 1: Accuracy and token-length changes of CAST relative to DAPO. The origin denotes the DAPO baseline. Each arrow points from DAPO to CAST; moving upward means higher accuracy, and moving right means fewer generated tokens.

1 INTRODUCTION

Recent advances in reinforcement learning with verifiable rewards (Shao et al., 2024; Guo et al., 2025a; Yu et al., 2025; Hou et al., 2025b; Liu et al., 2025) have quickly established themselves

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.

as a powerful paradigm for improving mathematical reasoning. However, the push toward longer reasoning chains has exposed a basic inefficiency: standard RL methods do not budget rollout compute and training signals where they matter most. Prior work (Sun et al., 2025a; Li et al., 2026c; Ning et al., 2025) reveals that different steps in a reasoning trace contribute very unevenly to the final answer. Answer-convergence and early-exit studies (Liu & Wang, 2025; Yang et al., 2025a) show that correctness is often decided by just a few pivotal steps, while many subsequent tokens are generated after the solution is already fixed. These findings point to two coupled challenges for reasoning RL: where to spend rollout compute during sampling, and how to assign credit to intermediate steps once trajectories are collected.

This inefficiency plays out differently on hard versus easy problems. On hard problems, early mistakes often lead the model toward an incorrect solution, and later self-reflection rarely helps: a recent analysis of long-CoT rollouts (Kang et al., 2025) finds that corrective reflections account for less than 2% of reflection transitions in most evaluated settings. This does not mean reflection is useless; rather, current RL methods fail to focus rollout compute and learning signals on the branch points where reflection can still make a difference. As a result, later tokens often just rationalize flawed reasoning instead of finding the right path. On mathematical reasoning tasks, models often converge to their final answers after roughly 60% of the reasoning steps (Liu & Wang, 2025), yet continue generating redundant checks and reflections (Yang et al., 2025a; Huang et al., 2026). In both cases, rollout budget and training signals are spent on steps that no longer affect the outcome. Simply adding a length penalty does not fix this: it punishes helpful reasoning steps along with redundant ones, ultimately hurting accuracy (Srivastava et al., 2025; Ning et al., 2025). What is needed is an RL framework that actively steers rollout compute toward unresolved branch points and assigns credit according to each step's real impact on the answer.

Existing reasoning RL methods do not fully address this need. Solution-level methods only reward complete responses based on final outcomes (Yu et al., 2025; Hou et al., 2025b; Liu et al., 2025), making it impossible to pinpoint which intermediate decisions actually mattered. Segment- and tree-based methods add intermediate structure (Guo et al., 2025b; Yang et al., 2025b; Hou et al., 2025a), but they typically branch using rigid schedules or simple heuristics (like fixed token intervals), and their credit estimates rely on local averages (like child success rates) rather than the full tree topology. As a result, they spend rollout budget on uninformative branches and miss the global structure of the sampled tree when assigning credit.

In this paper, we propose CAST, a tree-structured framework for long-chain reasoning RL. CAST organizes on-policy rollouts into selective reasoning trees, continuing exploration only at unresolved branch points where sampled continuations yield both correct and incorrect outcomes. It then pools all leaf rewards and the tree topology to estimate intermediate step values via a Gaussian-tree covariance model, deriving principled edge-level advantages. Figure 1 highlights the main outcome: compared with DAPO, CAST improves accuracy while generating substantially fewer tokens. Our main contributions are:

- **Selective tree exploration**: We introduce a selective reasoning-tree construction method that concentrates rollout compute strictly on unresolved branches with mixed outcomes.
- **Topology-aware credit assignment**: We develop a closed-form value estimator that uses a Gaussian-tree covariance model to pool information across all nodes and leaves, providing principled edge advantages instead of noisy local averages.
- **Comprehensive validation and efficiency gains**: Across five diverse backbones (1.7B to 35B MoE), CAST achieves a clear Pareto improvement, boosting accuracy while cutting response length by up to 25.8% and delivering a $1.8\times$ training speedup on Qwen3-4B-Base. Behavioral analyses further reveal a $3.8\times$ increase in the successful reflection rate on challenging problems.

2 Related Work

Outcome-supervised reinforcement learning with verifiable rewards (Shao et al., 2024; Guo et al., 2025a; Yu et al., 2025; Hou et al., 2025b; Liu et al., 2025) has substantially improved mathematical reasoning. However, as reasoning traces grow longer, allocating rollout compute and learning signal

becomes increasingly critical. We review two primary lines of work: credit assignment along linear trajectories and over structured reasoning trees.

Credit Assignment on Linear Trajectories. Several methods aim to distribute training signals along linear reasoning paths. VinePPO (Kazemnejad et al., 2024) estimates prefix values using Monte Carlo rollouts from intermediate states rather than relying on a learned critic. KTAE (Sun et al., 2025a) and OAR (Li et al., 2026c) redistribute sparse outcome rewards to individual tokens by measuring statistical association or outcome sensitivity. Other works localize training signals through generative step-wise critiques (Xie et al., 2025), counterfactual corrective interventions (Yang et al., 2026), or explicit step-level value estimation (Li et al., 2026b). However, these approaches operate largely post hoc: they refine credit after complete linear trajectories have already been sampled. Consequently, they provide limited control over rollout compute during sampling, meaning substantial exploration budget may be wasted on uninformative reasoning paths.

Credit Assignment on Reasoning Trees. A second line of work introduces tree structures directly into rollout sampling to capture alternative reasoning branches. TreeRPO (Yang et al., 2025b) and TreeRL (Hou et al., 2025a) construct tree rollouts to expose alternative intermediate steps and derive denser process-level supervision. Segment Policy Optimization (Guo et al., 2025b) assigns advantages to multi-token segments rather than entire responses, reusing shared prefixes through tree rollouts for long-chain reasoning. While tree search has also been widely explored for test-time scaling during inference (Li, 2025), applying tree rollouts to RL training presents distinct challenges. In current training-time tree methods, expansion is typically governed by rigid schedules or heuristics, such as branching at fixed token intervals (Guo et al., 2025b) or high-entropy positions (Hou et al., 2025a). Consequently, these heuristics may fail to identify which branch points are truly unresolved and would actually benefit from further exploration. Furthermore, they estimate node values from local statistics—such as averaging immediate child success rates—rather than modeling the full tree topology and pooling all observed leaves.

Overall, prior work often treats rollout sampling and credit assignment in isolation. CAST unifies them: it concentrates rollout compute on unresolved branch points during sampling, and leverages the full tree topology to assign fine-grained credit across all intermediate decisions.

3 METHODOLOGY

We describe CAST in three stages, following Figure 2: (1) **Selective tree construction** organizes on-policy rollouts into reasoning trees and expands only unresolved branch points whose sampled continuations contain both correct and incorrect outcomes. (2) **Topology-aware credit assignment** pools all observed leaf rewards and the hierarchical tree covariance to estimate intermediate node values in closed form. (3) **Tree-to-batch policy update** converts selected tree edges and contrastive suffix pairs into DAPO-compatible training samples.

3.1 SELECTIVE REASONING TREE CONSTRUCTION

Tree representation and node status. For each prompt x, CAST builds a rooted reasoning tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ with maximum depth D, where $\mathrm{depth}(n)$ is the path length from root n_{root} (containing prompt x, depth 0) to node n. Each non-root node stores one contiguous reasoning segment, so a root-to-leaf path represents one sampled trace split into at most D segments. The prefix q_n concatenates x with all segments on the path to n.

After sampled paths are inserted into the tree, each leaf ℓ receives a verifier reward $\tilde{r}_\ell \in \{0, 1\}$, indicating whether the final answer on that path is correct. Let $\mathcal{L}(n)$ denote the sampled descendant leaves under node n in the current tree. We define an outcome-consistency indicator for each node,

$$s(n) = \mathbb{I}[\forall \ell, \ell' \in \mathcal{L}(n),\ \tilde{r}_\ell = \tilde{r}_{\ell'}]\,, \tag{1}$$

where $\mathbb{I}[\cdot]$ is the indicator function. A node is *resolved* ($s(n) = 1$) if all descendant leaves share the same outcome (all correct or all incorrect), and *unresolved* ($s(n) = 0$) if outcomes are mixed. CAST expands only unresolved nodes with mixed sampled outcomes, while preserving all-correct prefixes for suffix length regularization.

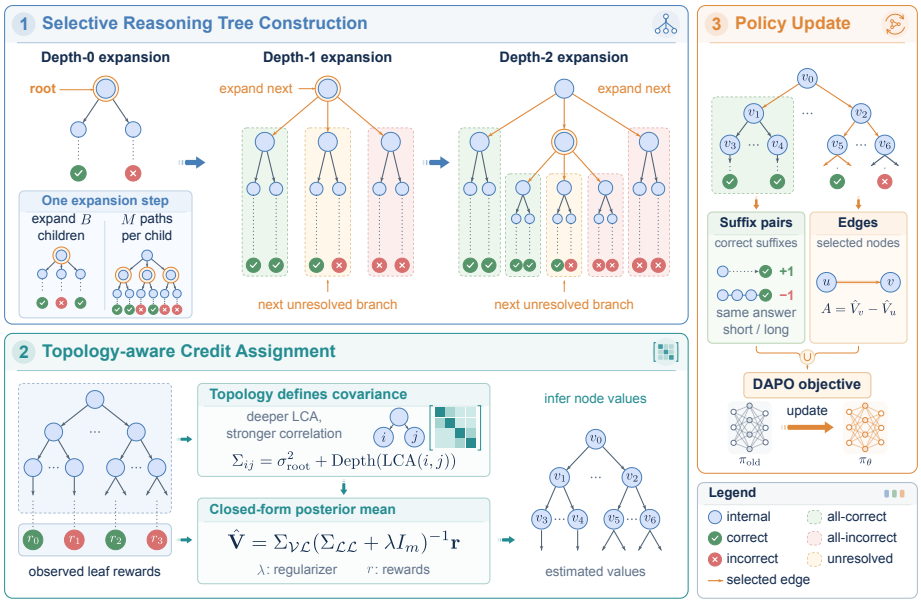

Figure 2: **Overview of CAST.** (1) Selective rollout builds a reasoning tree by expanding unresolved branch points and collecting leaf rewards. (2) Topology-aware credit assignment estimates node values from the completed tree via a closed-form Gaussian tree model. (3) Policy update converts selected edges and contrastive suffix pairs into DAPO training samples.

How a selected node is expanded. Given a selected node n at depth d, CAST reuses its existing sampled paths and adds continuations from prefix q_n until n has B children. Each added continuation is split post hoc into $D - d$ contiguous segments of approximately equal token count and inserted as a path in the tree. For each nonterminal child, additional continuations are sampled only until its subtree contains M descendant leaves; terminal children require no further completion. Here, B controls local branching width and M controls the completion budget per nonterminal child, with $B \geq M$.

How nodes are selected for expansion. CAST first probes the root by sampling M continuations from n_{root}. If initial rollouts are already all-correct or all-incorrect ($s(n_{\text{root}}) = 1$), expansion terminates immediately, skipping prompts with little learning signal. Otherwise, the root becomes the first selected node $n^{(0)} = n_{\text{root}}$. Let $\text{Ch}(n)$ denote the children of node n, and let $n^{(d)}$ denote the selected node at expansion round d. After expanding node $n^{(d)}$, CAST chooses the next node by uniformly sampling from its unresolved children:

$$\mathcal{U}_{d+1} = \{c \in \text{Ch}(n^{(d)}) : s(c) = 0,\ \text{depth}(c) < D\}, \qquad n^{(d+1)} \sim \text{Uniform}(\mathcal{U}_{d+1}) \ \text{ if } \mathcal{U}_{d+1} \neq \emptyset. \tag{2}$$

If $\mathcal{U}_{d+1} = \emptyset$, expansion stops. Thus, rollout follows a single unresolved chain, expanding strictly one node per depth. The complete rollout schedule is summarized in Algorithm 1 in Appendix A.1.

3.2 Topology-aware Credit Assignment

Given observed leaf rewards and the tree topology, the credit-assignment stage estimates latent values for all intermediate nodes, providing fine-grained advantage signals for policy optimization.

Gaussian tree model. CAST treats the value of each node as an unobserved scalar. For a constructed tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$, let $V(n) \in \mathbb{R}$ denote the latent value of node n. We assume that the root node follows a Gaussian distribution,

$$V(n_{\text{root}}) \sim \mathcal{N}(0, \sigma^2_{\text{root}}), \tag{3}$$

where $\sigma^2_{\text{root}} > 0$ is the root prior variance. Each directed edge $(n, c) \in \mathcal{E}$, from parent n to child c, contributes an independent Gaussian increment,

$$V(c) = V(n) + \epsilon_{(n,c)}, \qquad \epsilon_{(n,c)} \sim \mathcal{N}(0, 1). \tag{4}$$

Under this random walk, two nodes naturally have higher covariance when they share a longer prefix path:

$$\mathrm{Cov}(V(n_i), V(n_j)) = \sigma^2_{\mathrm{root}} + \mathrm{depth}(\mathrm{LCA}(n_i, n_j)). \tag{5}$$

Here $\mathrm{LCA}(n_i, n_j)$ denotes the lowest common ancestor of n_i and n_j. Through this covariance rule, the hierarchical tree topology directly dictates the statistical coupling across all node pairs.

Topology-aware value estimation. Let $\mathcal{V}$ denote all nodes and $\mathcal{L} = \{\ell_1, \ldots, \ell_m\}$ denote all leaves of $\mathcal{T}$, collecting their rewards into $\mathbf{r} = (\tilde{r}_{\ell_1}, \ldots, \tilde{r}_{\ell_m})^\top \in \mathbb{R}^m$. Under the tree covariance model, the regularized linear minimum mean-square error (LMMSE) estimate for all latent node values $\mathbf{V}$ is given in closed form:

$$\hat{\mathbf{V}} = \Sigma_{\mathcal{V}\mathcal{L}} \left(\Sigma_{\mathcal{L}\mathcal{L}} + \lambda I_m\right)^{-1} \mathbf{r}, \tag{6}$$

where $\lambda > 0$ is a ridge regularizer for numerical stability ($\lambda = 10^{-8}$ by default) and I_m is the $m \times m$ identity matrix. The covariance blocks $\Sigma_{\mathcal{V}\mathcal{L}} \in \mathbb{R}^{|\mathcal{V}| \times m}$ and $\Sigma_{\mathcal{L}\mathcal{L}} \in \mathbb{R}^{m \times m}$ are filled entrywise via Eq. 5: $[\Sigma_{\mathcal{V}\mathcal{L}}]_{n,j} = \mathrm{Cov}(V(n), V(\ell_j))$ and $[\Sigma_{\mathcal{L}\mathcal{L}}]_{i,j} = \mathrm{Cov}(V(\ell_i), V(\ell_j))$. Appendix A.2 provides the full derivation.

3.3 Tree-to-Batch Policy Update

Tree-to-batch conversion. Once node values are estimated, CAST converts selected tree decisions into DAPO-compatible samples. For each outgoing edge (u, v) of an expanded node, we assign edge advantage $A_{u \to v} = \hat{V}(v) - \hat{V}(u)$ and create $(q_u, y_{u \to v}, A_{u \to v})$, where q_u is the parent prefix and $y_{u \to v}$ is the segment on the edge. While all leaves inform value estimation, only selected outgoing edges form the edge batch $\mathcal{B}_{\mathrm{edge}}$.

To suppress redundant verification on problems the model already solves and mitigate the overthinking identified in Section 1, CAST extracts contrastive suffix pairs from all-correct internal nodes: the shortest and longest sampled suffixes $(\tau_{\mathrm{short}}, \tau_{\mathrm{long}})$ under node b receive $+1$ and -1 advantages in $\mathcal{B}_{\mathrm{suf}}$. Edge and suffix advantages are normalized separately within each tree. Suffix samples are capped at 10% of the combined sample count. The combined policy batch is $\mathcal{B} = \mathcal{B}_{\mathrm{edge}} \cup \mathcal{B}_{\mathrm{suf}}$ (see Appendix A.3).

DAPO policy update. We use DAPO (Yu et al., 2025) as the base update rule. For each sample $(q_i, y_i, A_i) \in \mathcal{B}$, q_i is the prefix, y_i is the generated segment or suffix, and A_i is the advantage assigned by the tree-to-batch conversion above. CAST changes how training samples are selected and how their advantages are constructed, while the clipped DAPO objective is unchanged; we provide the full objective in Appendix A.4.

4 Experiments

Our experiments evaluate whether CAST improves long-chain reasoning accuracy while substantially reducing generation and training costs across diverse model scales. We first compare overall reasoning accuracy across five diverse backbones, then examine error recovery and continued generation after an answer can already be elicited. Finally, ablation studies isolate how selective expansion and topology-aware credit assignment each contribute to the gains.

4.1 Experimental settings

We train CAST on five backbones: three Qwen3 models (Qwen Team, 2025) (Qwen3-1.7B-Base, Qwen3-4B-Base, and Qwen3-4B), DASD-4B-Thinking, and Qwen3.5-35B-A3B-Base (Qwen Team, 2026). DASD-4B-Thinking (Yan et al., 2026) is distilled from gpt-oss-120b (OpenAI, 2025b) with Qwen3-4B-Instruct-2507 as the student model. For training data, Qwen3-1.7B-Base and Qwen3-4B-Base use DAPO-Math-17k (Yu et al., 2025). Qwen3-4B, DASD-4B-Thinking, and Qwen3.5-35B-A3B-Base use DAPO-Math-17k together with NuminaMath-1.5-RL-Verifiable (nlile, 2025).

For training-time generation, we use an 8K-token budget for Qwen3-1.7B-Base and Qwen3-4B-Base, and a 16K-token budget for Qwen3-4B, DASD-4B-Thinking, and Qwen3.5-35B-A3B-Base. We

Table 1: Accuracy (%) across four Qwen backbones. DAPO (matched budget) increases sampling to match CAST's GPU time per training step. † denotes results reported by VCRL (Jiang et al., 2025), DiPO (Li et al., 2026a), PPPO (Sun et al., 2025b), and SSVPO (Li et al., 2026b).

Model / method	AIME24	AIME25	MATH500	Average
Qwen3-1.7B-Base				
Initial	3.33 ± 0.22	2.92 ± 0.22	55.74 ± 0.32	20.66
DAPO	15.42 ± 0.45	9.79 ± 0.39	73.73 ± 0.29	32.98
CAST	$\mathbf{17.29} \pm 0.47$	$\mathbf{11.46} \pm 0.42$	$\mathbf{74.61} \pm 0.28$	**34.45**
Qwen3-4B-Base				
Initial	6.46 ± 0.32	4.79 ± 0.28	40.22 ± 0.31	17.16
DAPO	25.05 ± 0.49	21.20 ± 0.61	84.69 ± 0.28	43.65
DAPO (matched budget)	26.67 ± 0.47	23.07 ± 0.59	86.80 ± 0.26	45.51
VCRL†	23.96	22.71	86.48	44.38
DiPO†	29.17	24.58	87.00	46.92
TreeRPO	26.82 ± 0.69	22.92 ± 0.64	86.13 ± 0.19	45.29
TreeRL	28.12 ± 0.53	23.65 ± 0.50	86.52 ± 0.24	46.10
CAST	$\mathbf{30.10} \pm 0.60$	$\mathbf{25.42} \pm 0.45$	$\mathbf{87.51} \pm 0.20$	**47.68**
Qwen3-4B				
Initial	62.92 ± 0.55	49.38 ± 0.61	94.80 ± 0.30	69.03
DAPO	66.04 ± 0.54	57.08 ± 0.61	94.50 ± 0.23	72.54
PPPO†	63.54	53.44	94.60	70.53
SSVPO†	66.60	61.10	95.00	74.23
CAST	$\mathbf{67.81} \pm 0.67$	$\mathbf{62.08} \pm 0.78$	$\mathbf{95.56} \pm 0.18$	**75.15**
Qwen3.5-35B-A3B-Base / separate benchmark suite				
Model / method	AIME26	APEX25	HMMT-Feb26	Average
DAPO	83.02 ± 0.75	21.35 ± 0.74	68.94 ± 1.08	57.77
CAST	$\mathbf{87.71} \pm 0.51$	$\mathbf{26.63} \pm 1.00$	$\mathbf{73.30} \pm 0.87$	**62.55**

Table 2: Accuracy (%) on DASD-4B-Thinking with 16K and 32K evaluation token budgets.

Model / method	AIME24	AIME25	MATH500	APEX25	Average
DASD-4B-Thinking / 16K evaluation budget					
Initial	63.13 ± 1.91	51.04 ± 1.60	93.48 ± 0.55	2.08 ± 0.63	52.43
DAPO	68.54 ± 1.72	58.33 ± 1.72	94.19 ± 0.43	7.16 ± 0.74	57.06
CAST	$\mathbf{71.46} \pm 1.55$	$\mathbf{62.08} \pm 1.85$	$\mathbf{95.08} \pm 0.39$	$\mathbf{10.03} \pm 0.77$	**59.66**
DASD-4B-Thinking / 32K evaluation budget					
Initial	75.83 ± 1.37	66.67 ± 1.62	95.09 ± 0.49	7.81 ± 0.80	61.35
CAST	$\mathbf{76.88} \pm 1.23$	$\mathbf{67.71} \pm 1.59$	$\mathbf{96.28} \pm 0.57$	$\mathbf{12.24} \pm 0.71$	**63.28**

retain these budgets at evaluation, except that DASD-4B-Thinking is also evaluated with a 32K-token budget. Experiments on the 1.7B and 4B backbones used two nodes, each with eight A100 GPUs, while the 35B experiment used four nodes. For the experiments we conduct, evaluation uses matched context budgets and identical protocols. Further details on optimization, sampling, and tree construction are provided in Appendix B.1.

Our primary benchmarks are AIME24 and AIME25 (Mathematical Association of America, 2026) and MATH500 (Hendrycks et al., 2021; Lightman et al., 2024); DASD is also evaluated on MathArena Apex (APEX25; Dekoninck et al., 2025). For the stronger 35B backbone, we use AIME26, APEX25, and HMMT-Feb26 (Harvard-MIT Mathematics Tournament, 2026). For our evaluations in Table 1, accuracy is averaged over 16 sampled responses per problem on MATH500 and 64 on the other benchmarks. Table averages give equal weight to each benchmark. Reported $\pm$ values indicate standard errors; † marks results taken from prior work.

4.2 Main results

Accuracy across backbones. CAST improves upon DAPO on every reported benchmark for both base models (Table 1), with average gains of 1.47 points on Qwen3-1.7B-Base and 4.03 points on Qwen3-4B-Base. On Qwen3-4B-Base, CAST reaches 47.68% average accuracy, outperforming TreeRPO by 2.39 points and TreeRL by 1.58 points across all three benchmarks. While TreeRPO

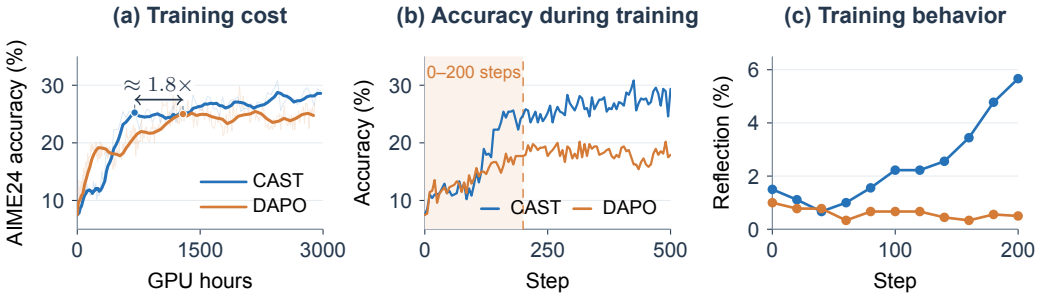

Figure 3: Training efficiency and dynamics on Qwen3-4B-Base. AIME24 accuracy versus (a) training GPU hours and (b) training steps; shaded: steps 0–200 shown in (c). (c) Proportion of reflection-labeled segments during training; each complete response is divided into five segments.

branches uniformly with noisy local child-mean credit and TreeRL branches on model entropy, CAST leverages verifier-guided outcome consistency and global covariance pooling to achieve higher benchmark accuracy. CAST also exceeds the strongest reported external baseline, DiPO, by 0.76 points. On the reasoning model Qwen3-4B, it reaches 75.15%, compared with 72.54% for DAPO and 74.23% for SSVPO.

On the larger Qwen3.5-35B-A3B-Base backbone, CAST increases average accuracy from 57.77% to 62.55% on the more challenging benchmark suite. It improves over DAPO by 4.69 points on AIME26, 5.28 points on APEX25, and 4.36 points on HMMT-Feb26.

The gains also extend to the distilled reasoning model DASD-4B-Thinking (Table 2). At a 16K evaluation budget, CAST improves average accuracy over DAPO from 57.06% to 59.66%, a 2.60-point gain, and performs better on all four benchmarks. At 32K, it improves over the initial distilled checkpoint from 61.35% to 63.28%. Thus, CAST remains effective when training starts from a strong reasoning checkpoint.

Training efficiency. On Qwen3-4B-Base, although tree construction samples more rollout tokens per step, CAST's concentrated non-zero advantage signals yield substantially higher learning efficiency. As a result, CAST requires fewer parameter updates to reach target accuracy, overtaking DAPO and achieving an approximately $1.8\times$ wall-clock speedup in GPU hours on AIME24 (Figure 3a).

4.3 Behavioral Analysis

To understand how CAST achieves higher accuracy with shorter responses, we examine two reasoning inefficiencies in baseline methods. We analyze error recovery in evaluation and training, and how accuracy gains vary with problem difficulty. We then measure continued generation after the final answer can already be elicited to identify where token savings arise. Unless otherwise stated, analyses and ablations below use Qwen3-4B-Base.

4.3.1 Error Recovery and Gains Across Problem Difficulty

We assess error recovery on AIME24 using segment-level annotations. We split each evaluation trajectory into eight equal-length segments and use GPT-5.2 (OpenAI, 2025a) to annotate critical errors, reflection, and successful reflection with the prompts in Appendix B.3. We label a reflection as successful (SR) if it redirects an incorrect trajectory to the ground-truth final answer. As shown in Table 4, CAST reduces the critical-error rate from 35.2% to 27.6% and increases the reflection rate from 20.8% to 23.6%. Crucially, the successful reflection rate conditional on reflection rises from 4.8% to 18.3%, representing a $3.8\times$ increase. Appendix C.1 illustrates this behavior qualitatively.

To understand how this recovery advantage develops during training, we examine the learning curves in Figure 3(b). CAST's accuracy rises sharply between steps 100 and 200, separating from DAPO. Figure 3(c) shows that this accuracy transition coincides with a rise in CAST's reflection rate, while DAPO's remains low (where each response is divided into five segments for reflection annotation). Together with the evaluation results in Table 4, this pattern suggests that selectively expanding unresolved branches enables the policy to learn from corrective continuations, thereby improving error recovery and driving this performance gain.

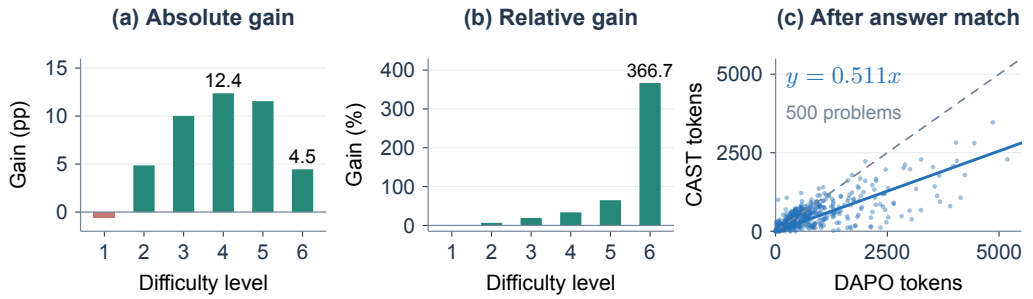

Figure 4: Gains across problem difficulty and response efficiency on MATH500. (a,b) Absolute and relative accuracy gains of CAST over DAPO across six difficulty bins, from easiest to hardest. (c) Per-problem mean tokens after the first answer match; dashed: $y = x$, solid: fit through the origin.

Table 3: Mean Qwen3 token usage on AIME24, AIME25, and MATH500.

Model	DAPO	CAST	Change
1.7B-Base	3506	**2779**	−20.7%
4B-Base	3385	**2513**	−25.8%
4B	9486	**8301**	−12.5%

Table 4: AIME24 reflection: error and reflection rates over all annotated segments (SR: successful reflection).

Metric	DAPO	CAST	Rel. change
Critical error rate	35.2%	27.6%	↓**21.6%**
Reflection rate	20.8%	23.6%	↑13.5%
SR | reflection	4.8%	18.3%	↑**281.3%**

To identify which problems benefit most, we group MATH500 problems into six equal-width bins of combined DAPO and CAST accuracy. Figures 4(a,b) show that the performance gains of CAST concentrate heavily on challenging problems. In the hardest difficulty bin, where standard rollouts rarely succeed, CAST lifts accuracy from 1.2% to 5.7%, achieving a 366.7% relative increase. Together with the error recovery analysis, these findings show that CAST improves performance on problems where baseline methods struggle, supporting the efficacy of targeted exploration and credit assignment on demanding tasks.

4.3.2 Locating token savings and mitigating redundancy

Across the three Qwen3 backbones, CAST improves accuracy while generating 12.5–25.8% fewer tokens than DAPO (Tables 1 and 3). To locate where these savings arise, we follow answer-convergence analysis (Liu & Wang, 2025) and truncate each MATH500 response at 10%, 20%, ..., 90% of its length, prompting for an immediate final answer. For each response, we count the tokens after the earliest tested cutoff whose elicited answer matches the final answer of the complete response. Comparing per-problem tokens between CAST and DAPO, most points fall below the diagonal in Figure 4(c), with a linear fit through the origin yielding a slope of 0.51. These results indicate that CAST reduces continued generation after the final answer can already be elicited.

To foster this concise behavior during training, CAST extracts contrastive suffix pairs at all-correct nodes to favor shorter continuations among trajectories with identical verified outcomes. Unlike standard length penalties that uniformly penalize entire sequences and risk corrupting essential reasoning steps, this mechanism leaves the verified reasoning prefix untouched and applies regularizing gradients exclusively to redundant post-solution suffixes. Holding tree construction and value estimation fixed, incorporating these suffix samples on Qwen3-4B-Base reduces mean response length from 3265 to 2513 tokens (a 23.0% reduction) while maintaining nearly identical average accuracy (47.68% vs. 47.75% without suffix comparison; Table 8; Appendix C.2).

4.4 Mechanism and ablation studies

Selective expansion retains deeper training signal. With equally sized trees, CAST yields a larger fraction of non-zero advantage segments than TreeRPO at every depth, reaching 21.57% versus 4.99% at depth 5 in Figure 5(a). To assess node selection, we compare against a greedy baseline that expands children with highest estimated value rather than targeting unresolved uncertainty. Across 128 problems, prioritizing unresolved nodes yields substantially more non-zero-advantage samples

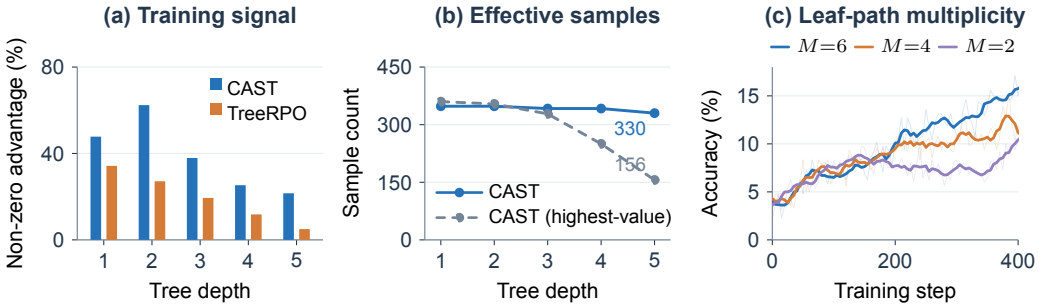

Figure 5: Training signal and leaf-path multiplicity. (a) Fraction of segments with non-zero advantage by tree depth for equally sized CAST and TreeRPO trees. (b) Effective samples under unresolved and highest-value expansion. (c) Validation accuracy for $M = 2, 4, 6$ leaf paths.

Table 5: Node-value MSE and Gaussian-tree sensitivity (*: training default).

Estimator	**MSE ↓**	**Reduction**
Child-mean	0.02371	–
CAST	**0.01843**	**22.3%**

Root variance		**Ridge parameter**	
σ^2_{root}	MSE ↓	λ	MSE ↓
1	0.01933	$10^{-8\,*}$	0.01843
10	0.01850	0.01	0.01843
100	0.01844	0.1	0.01838
1000*	**0.01843**	1	**0.01831**

Table 6: Component ablations on Qwen3-4B-Base (accuracy ± standard error).

Configuration	**AIME24**	**AIME25**	**MATH500**
DAPO	25.05 ± 0.49	21.20 ± 0.61	84.69 ± 0.28
CAST (child-mean)	28.84 ± 0.57	24.71 ± 0.48	86.17 ± 0.24
CAST (highest-value)	27.48 ± 0.54	24.73 ± 0.50	85.98 ± 0.24
CAST	$\mathbf{30.10} \pm 0.60$	$\mathbf{25.42} \pm 0.45$	$\mathbf{87.51} \pm 0.20$

from depth 3 onward: 330 versus 156 at depth 5 in Figure 5(b). Because greedy expansion repeatedly explores paths already solved reliably, it starves deeper nodes of contrastive feedback and incurs a 1.61-point drop in average accuracy in Table 6. This confirms that expanding unresolved branches sustains critical training signals throughout deep rollouts and improves downstream accuracy.

Topology-aware values improve the credit signal. To test whether topology improves credit assignment, we compare Gaussian-tree and child-mean values on 64 CAST trees from MATH500. Each internal node's reference accuracy is estimated from 200 independent continuations. Gaussian-tree MSE is 0.01843 versus 0.02371 for child-mean, a 22.3% reduction in Table 5. These values define edge advantages through child–parent differences. In Table 6, comparing DAPO at 43.65%, CAST with child-mean at 46.57%, and full CAST at 47.68% isolates the core components: selective rollout adds +2.92 points under local credit, while global topology-aware credit adds another +1.11 points across all benchmarks. These results support the benefits of combining selective allocation with topology-aware credit. Additionally, sensitivity sweeps in Table 5 confirm that the estimator maintains stable MSE across eight orders of magnitude of the ridge regularizer and broad ranges of root variance.

Effect of leaf-path multiplicity on learning. Since node resolution depends on sampled outcomes, we vary the leaf-path multiplicity M on Qwen3-1.7B-Base. In later training steps, $M = 6$ achieves higher validation accuracy than $M = 4$ or $M = 2$ in Figure 5(c), confirming that denser leaf sampling provides more reliable exploration signals during training.

5 Conclusion

To address premature failures on complex tasks and redundant overthinking on solved steps, we have presented CAST, a framework that unifies selective tree exploration with topology-aware credit assignment. By concentrating rollout compute on unresolved branch points and deriving intermediate step values from the full tree topology, CAST delivers fine-grained learning signals to decisive reasoning steps while naturally eliminating wasteful verification. Across five diverse backbones, CAST consistently improves accuracy, boosts the successful reflection rate by 3.8×, and cuts response length by up to 25.8% with a 1.8× training speedup.

AI USE STATEMENT

We used large language models to polish the manuscript and annotate reasoning trajectories for critical errors, reflection, and successful reflection in our experiments.

REFERENCES

Jasper Dekoninck, Nikola Jovanović, Ivo Petrov, and Martin Vechev. MathArena Apex: Unconquered final-answer problems. MathArena Blog, 2025. URL https://matharena.ai/apex/. Accessed: 2026-05-07.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645:633–638, 2025a. doi: 10.1038/s41586-025-09422-z. URL https://www.nature.com/articles/s41586-025-09422-z.

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. *arXiv preprint arXiv:2505.23564*, 2025b. URL https://arxiv.org/abs/2505.23564.

Harvard-MIT Mathematics Tournament. Archive of February 2026, 2026. URL https://www.hmmt.co/www/archive/292. Accessed: 2026-09-20.

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. *arXiv preprint arXiv:2103.03874*, 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. *arXiv preprint arXiv:2506.11902*, 2025a. URL https://arxiv.org/abs/2506.11902.

Zhenyu Hou, Xin Lv, Rui Lu, Jiajie Zhang, Yujiang Li, Zijun Yao, Juanzi Li, Jie Tang, and Yuxiao Dong. T1: Advancing language model reasoning through reinforcement learning and inference scaling. *arXiv preprint arXiv:2501.11651*, 2025b. URL https://arxiv.org/abs/2501.11651.

Zixuan Huang, Xin Xia, Yuxi Ren, Jianbin Zheng, Xuanda Wang, Zhixia Zhang, Hongyan Xie, Songshi Liang, Zehao Chen, Xuefeng Xiao, Fuzhen Zhuang, Jianxin Li, Yikun Ban, and Deqing Wang. Does your reasoning model implicitly know when to stop thinking? *arXiv preprint arXiv:2602.08354*, 2026. URL https://arxiv.org/abs/2602.08354.

Guochao Jiang, Wenfeng Feng, Guofeng Quan, Chuzhan Hao, Yuewei Zhang, Guohua Liu, and Hao Wang. Vcrl: Variance-based curriculum reinforcement learning for large language models. *arXiv preprint arXiv:2509.19803*, 2025. URL https://arxiv.org/abs/2509.19803.

Liwei Kang, Yue Deng, Yao Xiao, Zhanfeng Mo, Wee Sun Lee, and Lidong Bing. First try matters: Revisiting the role of reflection in reasoning models. *arXiv preprint arXiv:2510.08308*, 2025. URL https://arxiv.org/abs/2510.08308.

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. *arXiv preprint arXiv:2410.01679*, 2024. URL https://arxiv.org/abs/2410.01679.

Xiaofan Li, Ming Yang, Zhiyuan Ma, Shichao Ma, Jintao Du, Yu Cheng, Weiqiang Wang, Zhizhong Zhang, Xin Tan, Yanyun Qu, Lizhuang Ma, and Yuan Xie. Dipo: Disentangled perplexity policy optimization for fine-grained exploration-exploitation trade-off. *arXiv preprint arXiv:2604.13902*, 2026a. URL https://arxiv.org/abs/2604.13902.

Yang Li. Policy guided tree search for enhanced llm reasoning. *arXiv preprint arXiv:2502.06813*, 2025. URL https://arxiv.org/abs/2502.06813.

Yugu Li, Zehong Cao, Jianglin Qiao, and Siyi Hu. Ssvpo: Effective step-level credit assignment for rl training of language models. In *International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=g33DGvnHYd.

Ziheng Li, Liu Kang, Feng Xiao, Luxi Xing, Qingyi Si, Zhuoran Li, Weikang Gong, Deqing Yang, Yanghua Xiao, and Hongcheng Guo. Outcome-grounded advantage reshaping for fine-grained credit assignment in mathematical reasoning. *arXiv preprint arXiv:2601.07408*, 2026c. URL https://arxiv.org/abs/2601.07408.

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://openreview.net/forum?id=v8L0pN6EOi.

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

Xin Liu and Lu Wang. Answer convergence as a signal for early stopping in reasoning. *arXiv preprint arXiv:2506.02536*, 2025. URL https://arxiv.org/abs/2506.02536.

Mathematical Association of America. MAA Invitational Competitions, 2026. URL https://maa.org/maa-invitational-competitions/. Accessed: 2026-05-07.

Yansong Ning, Wei Li, Jun Fang, Naiqiang Tan, and Hao Liu. Not all thoughts are generated equal: Efficient llm reasoning via multi-turn reinforcement learning. *arXiv preprint arXiv:2505.11827*, 2025. URL https://arxiv.org/abs/2505.11827.

nlile. Numinamath-1.5-rl-verifiable. Hugging Face Dataset Repository, 2025. URL https://huggingface.co/datasets/nlile/NuminaMath-1.5-RL-Verifiable.

OpenAI. Introducing gpt-5.2. OpenAI Product Release, 2025a. URL https://openai.com/index/introducing-gpt-5-2/. Accessed: 2026-04-27.

OpenAI. gpt-oss-120b & gpt-oss-20b Model Card. OpenAI Model Card, 2025b. URL https://openai.com/index/gpt-oss-model-card/. Accessed: 2026-05-07.

Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Qwen Team. Qwen3.5-35B-A3B-Base: Model card. Hugging Face, 2026. URL https://huggingface.co/Qwen/Qwen3.5-35B-A3B-Base. Accessed: 2026-09-20.

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. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

Gaurav Srivastava, Aafiya Hussain, Sriram Srinivasan, and Xuan Wang. Do llms overthink basic math reasoning? benchmarking the accuracy-efficiency tradeoff in language models. *arXiv preprint arXiv:2507.04023*, 2025. URL https://arxiv.org/abs/2507.04023.

Wei Sun, Wen Yang, Pu Jian, Qianlong Du, Fuwei Cui, Shuo Ren, and Jiajun Zhang. Ktae: A model-free algorithm to key-tokens advantage estimation in mathematical reasoning. *arXiv preprint arXiv:2505.16826*, 2025a. URL https://arxiv.org/abs/2505.16826.

Yiliu Sun, Zicheng Zhao, Yang Wei, Yanfang Zhang, and Chen Gong. Well begun, half done: Reinforcement learning with prefix optimization for llm reasoning. *arXiv preprint arXiv:2512.15274*, 2025b. URL https://arxiv.org/abs/2512.15274.

Guofu Xie, Yunsheng Shi, Hongtao Tian, Ting Yao, and Xiao Zhang. Capo: Towards enhancing llm reasoning through generative credit assignment. *arXiv preprint arXiv:2508.02298*, 2025. URL https://arxiv.org/abs/2508.02298.

Shaotian Yan, Kaiyuan Liu, Chen Shen, Bing Wang, Sinan Fan, Jun Zhang, Yue Wu, Zheng Wang, and Jieping Ye. Distribution-aligned sequence distillation for superior long-cot reasoning. *arXiv preprint arXiv:2601.09088*, 2026. URL https://arxiv.org/abs/2601.09088.

Chenxu Yang, Qingyi Si, Yongjie Duan, Zheliang Zhu, Chenyu Zhu, Qiaowei Li, Minghui Chen, Zheng Lin, and Weiping Wang. Dynamic early exit in reasoning models. *arXiv preprint arXiv:2504.15895*, 2025a. URL https://arxiv.org/abs/2504.15895.

Matthew Y. R. Yang, Hao Bai, Ian Wu, Gene Yang, Amrith Setlur, and Aviral Kumar. Int: Self-proposed interventions enable credit assignment in llm reasoning. *arXiv preprint arXiv:2601.14209*, 2026. URL https://arxiv.org/abs/2601.14209.

Zhicheng Yang, Zhijiang Guo, Yinya Huang, Xiaodan Liang, Yiwei Wang, and Jing Tang. Treerpo: Tree relative policy optimization. *arXiv preprint arXiv:2506.05183*, 2025b. 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. *arXiv preprint arXiv:2503.14476*, 2025. URL https://arxiv.org/abs/2503.14476.

A Additional method details

A.1 Selective reasoning tree rollout details

Algorithm 1 summarizes the selective rollout schedule in Section 3.1.

Algorithm 1 CAST Selective Tree Rollout Schedule

Input: Prompt x, policy π_θ, max depth D, branching factor B, leaf-path multiplicity M, with $B \geq M$.
Initialize tree $\mathcal{T} \leftarrow (\{n_{\text{root}}\}, \emptyset)$ with $q_{n_{\text{root}}} = x$, and selected edges $\mathcal{E}_{\text{sel}} \leftarrow \emptyset$.
Convention: AddRollout(u) samples a continuation from q_u, splits it into $D - \text{depth}(u)$ segments of approximately equal token count, inserts the path, and evaluates its terminal reward.
Call AddRollout(n_{root}) M times and compute $s(n_{\text{root}})$.
if $s(n_{\text{root}}) = 1$ **then return** completed tree $\mathcal{T}, \mathcal{E}_{\text{sel}}$ ▷ Skip all-correct or all-incorrect prompts
end if
Set current node $n \leftarrow n_{\text{root}}$.
while $\text{depth}(n) < D$ **do**
    Call AddRollout(n) while $|\text{Ch}(n)| < B$, retaining existing paths.
    for each child $c \in \text{Ch}(n)$ with $\text{depth}(c) < D$ **do**
        Call AddRollout(c) while $|\mathcal{L}(c)| < M$.
    end for
    Update consistency indicators on affected nodes and ancestors.
    $\mathcal{E}_{\text{sel}} \leftarrow \mathcal{E}_{\text{sel}} \cup \{(n, c) : c \in \text{Ch}(n)\}$.
    Candidate set $\mathcal{U} \leftarrow \{c \in \text{Ch}(n) : s(c) = 0, \text{depth}(c) < D\}$.
    if $\mathcal{U} = \emptyset$ **then break** ▷ No unresolved branches remain
    end if
    Sample $n \sim \text{Uniform}(\mathcal{U})$.
end while
return selective reasoning tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ and selected edges $\mathcal{E}_{\text{sel}}$.

A.2 Gaussian-tree covariance and LMMSE derivation

We derive the tree covariance in Appendix A.2.1 and the minimum-MSE linear estimator under the Gaussian tree model in Appendix A.2.2, yielding Eq. 6.

Gaussian tree model. We condition on a fixed constructed tree and model all node values as real-valued random variables. All expectations and covariances in this derivation are taken under the model specified below. Let n_{root} be the root node. We assume

$$V(n_{\text{root}}) \sim \mathcal{N}(0, \sigma^2_{\text{root}}), \tag{7}$$

and for each edge $(n, c) \in \mathcal{E}$,

$$V(c) = V(n) + \epsilon_{(n,c)}, \qquad \epsilon_{(n,c)} \sim \mathcal{N}(0, 1), \tag{8}$$

with edge increments that are mutually independent and independent of $V(n_{\text{root}})$. For an edge $e = (u, v)$, write $\epsilon_e := \epsilon_{(u,v)}$.

Let $\mathcal{P}(n)$ denote the set of edges on the path from the root to node n. By recursively expanding the edge relation along this path, every node value can be written as the root value plus the increments on its root-to-node path:

$$V(n) = V(n_{\text{root}}) + \sum_{e \in \mathcal{P}(n)} \epsilon_e. \tag{9}$$

A.2.1 TREE-TOPOLOGY COVARIANCE VIA LOWEST COMMON ANCESTORS

We first derive the covariance formula in Eq. 5. For two nodes n_i and n_j, substituting Eq. 9 gives

$$\mathrm{Cov}(V(n_i), V(n_j)) = \mathrm{Cov}\left(V(n_{\mathrm{root}}) + \sum_{e\in\mathcal{P}(n_i)} \epsilon_e,\ V(n_{\mathrm{root}}) + \sum_{e'\in\mathcal{P}(n_j)} \epsilon_{e'}\right) \tag{10}$$

$$= \mathrm{Var}(V(n_{\mathrm{root}})) + \sum_{e\in\mathcal{P}(n_i)} \sum_{e'\in\mathcal{P}(n_j)} \mathrm{Cov}(\epsilon_e, \epsilon_{e'}). \tag{11}$$

The cross-covariance terms between $V(n_{\mathrm{root}})$ and any edge increment vanish because the root value is independent of all edge increments. The covariance between two edge increments also vanishes unless the two edges are the same edge:

$$\mathrm{Cov}(\epsilon_e, \epsilon_{e'}) = \begin{cases} 1, & e = e', \\ 0, & e \neq e', \end{cases} \tag{12}$$

because edge increments are mutually independent and have unit variance. Hence only edges shared by both root-to-node paths contribute to the covariance:

$$\mathrm{Cov}(V(n_i), V(n_j)) = \mathrm{Var}(V(n_{\mathrm{root}})) + \sum_{e\in\mathcal{P}(n_i)\cap\mathcal{P}(n_j)} \mathrm{Var}(\epsilon_e) \tag{13}$$

$$= \sigma^2_{\mathrm{root}} + |\mathcal{P}(n_i) \cap \mathcal{P}(n_j)| \tag{14}$$

$$= \sigma^2_{\mathrm{root}} + \mathrm{depth}(\mathrm{LCA}(n_i, n_j)). \tag{15}$$

where $\mathrm{LCA}(n_i, n_j)$ denotes the lowest common ancestor of n_i and n_j. The final equality holds because $\mathcal{P}(n_i) \cap \mathcal{P}(n_j)$ is exactly the set of edges on the path from the root to $\mathrm{LCA}(n_i, n_j)$. The number of such edges is the depth of the lowest common ancestor. This proves Eq. 5, the covariance structure used by the Gaussian tree estimator in the main text.

A.2.2 DERIVATION OF THE TOPOLOGY-AWARE LINEAR ESTIMATOR

Let $\mathbf{V} \in \mathbb{R}^{|\mathcal{V}|}$ collect the values of all nodes in the constructed tree in a fixed order, and let $\mathcal{L} = \{\ell_1, \ldots, \ell_m\}$ be the leaf set. Write $\mathbf{V}_{\mathcal{L}} = (V(\ell_1), \ldots, V(\ell_m))^\top$ for the leaf values. We introduce a real-valued observation vector

$$\mathbf{Y} = \mathbf{V}_{\mathcal{L}} + \boldsymbol{\eta}, \qquad \boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0}, \lambda I_m), \tag{16}$$

where $\boldsymbol{\eta}$ is independent of $\mathbf{V}$ and $\lambda > 0$. The observation-noise variance λ yields the ridge term used to stabilize the covariance inversion. Since $\mathbf{V}$ and $\mathbf{Y}$ have zero mean, their second-order moments are

$$C_{\mathcal{V}Y} = \mathbb{E}[\mathbf{V}\mathbf{Y}^\top] = \Sigma_{\mathcal{V}\mathcal{L}}, \qquad C_{YY} = \mathbb{E}[\mathbf{Y}\mathbf{Y}^\top] = \Sigma_{\mathcal{L}\mathcal{L}} + \lambda I_m, \tag{17}$$

where I_m is the $m \times m$ identity matrix. The entries of the node–leaf and leaf–leaf covariance blocks are given by Eq. 5.

We now minimize MSE over linear estimators of all node values. Consider estimators of the form

$$\hat{\mathbf{V}} = A\mathbf{Y}, \tag{18}$$

where $A \in \mathbb{R}^{|\mathcal{V}|\times m}$ is a matrix to be chosen. The mean-square error objective is

$$\mathcal{L}(A) = \mathbb{E}\left[\|\mathbf{V} - A\mathbf{Y}\|_2^2\right]. \tag{19}$$

Expanding the squared norm and using $\|z\|_2^2 = z^\top z$ gives

$$\mathcal{L}(A) = \mathbb{E}\left[(\mathbf{V} - A\mathbf{Y})^\top(\mathbf{V} - A\mathbf{Y})\right] \tag{20}$$

$$= \mathbb{E}[\mathbf{V}^\top\mathbf{V}] - 2\,\mathbb{E}[\mathbf{V}^\top A\mathbf{Y}] + \mathbb{E}[\mathbf{Y}^\top A^\top A\mathbf{Y}]. \tag{21}$$

The first term does not depend on A. The second and third terms can be written with traces:

$$\mathbb{E}[\mathbf{V}^\top A\mathbf{Y}] = \mathrm{tr}(A\,C_{Y\mathcal{V}}) = \mathrm{tr}(A\Sigma_{\mathcal{L}\mathcal{V}})\,, \tag{22}$$

$$\mathbb{E}[\mathbf{Y}^\top A^\top A\mathbf{Y}] = \mathrm{tr}\big(A\,C_{YY}\,A^\top\big) = \mathrm{tr}\big(A(\Sigma_{\mathcal{L}\mathcal{L}} + \lambda I_m)A^\top\big)\,. \tag{23}$$

Therefore, up to the constant term $\mathbb{E}[\mathbf{V}^\top \mathbf{V}]$,

$$\mathcal{L}(A) = \text{const} - 2\,\text{tr}(A\Sigma_{\mathcal{L}\mathcal{V}}) + \text{tr}\big(A(\Sigma_{\mathcal{L}\mathcal{L}} + \lambda I_m)A^\top\big)\,. \tag{24}$$

Taking the derivative with respect to A gives

$$\frac{\partial \mathcal{L}(A)}{\partial A} = -2\Sigma_{\mathcal{V}\mathcal{L}} + 2A\left(\Sigma_{\mathcal{L}\mathcal{L}} + \lambda I_m\right). \tag{25}$$

Setting this derivative to zero gives the normal equation

$$A\left(\Sigma_{\mathcal{L}\mathcal{L}} + \lambda I_m\right) = \Sigma_{\mathcal{V}\mathcal{L}}. \tag{26}$$

Since $\lambda > 0$, the matrix $\Sigma_{\mathcal{L}\mathcal{L}} + \lambda I_m$ is positive definite and invertible. Thus the unique minimum-MSE linear estimator under this real-valued model has

$$A^\star = \Sigma_{\mathcal{V}\mathcal{L}}\left(\Sigma_{\mathcal{L}\mathcal{L}} + \lambda I_m\right)^{-1}. \tag{27}$$

For any observed leaf-target vector $\mathbf{r} \in \mathbb{R}^m$, evaluating this linear estimator at $\mathbf{Y} = \mathbf{r}$ yields

$$\hat{\mathbf{V}} = \Sigma_{\mathcal{V}\mathcal{L}}\left(\Sigma_{\mathcal{L}\mathcal{L}} + \lambda I_m\right)^{-1}\mathbf{r}, \tag{28}$$

In CAST, we use the verifier rewards $\mathbf{r} = (\tilde{r}_{\ell_1}, \ldots, \tilde{r}_{\ell_m})^\top$ as the numerical leaf targets in this model-derived estimator, giving exactly the formula in Section 3.2. For a single node n, the corresponding row of $\Sigma_{\mathcal{V}\mathcal{L}}$ gives

$$\hat{V}(n) = \Sigma_{n\mathcal{L}}\left(\Sigma_{\mathcal{L}\mathcal{L}} + \lambda I_m\right)^{-1}\mathbf{r}. \tag{29}$$

A.3 Batch construction and filtering details

For each all-correct internal node, we consider the shortest and longest sampled suffixes, with lengths L_{short} and L_{long}. Retained pairs satisfy

$$\text{Acc}_{\text{tree}} > 0.75, \qquad L_{\text{long}} - L_{\text{short}} \geq 200, \qquad \frac{L_{\text{long}}}{L_{\text{short}}} \geq 2.0, \qquad \rho_{\text{suf}} \leq 0.1. \tag{30}$$

Here, tree accuracy is the mean verifier reward over terminal leaves, and ρ_{suf} is the suffix sample fraction in the combined batch:

$$\text{Acc}_{\text{tree}} = \frac{1}{|\mathcal{L}|}\sum_{\ell \in \mathcal{L}} \tilde{r}_\ell, \qquad \rho_{\text{suf}} = \frac{|\mathcal{B}_{\text{suf}}|}{|\mathcal{B}_{\text{edge}}| + |\mathcal{B}_{\text{suf}}|}. \tag{31}$$

The 10% cap applies to sample counts.

Separate advantage normalization. Within each tree, edge and suffix advantages are standardized separately using their respective means and standard deviations before being combined for the policy update. We add $\epsilon = 10^{-8}$ to each standard deviation for numerical stability.

A.4 DAPO policy objective

We optimize the reasoning policy using the DAPO objective (Yu et al., 2025), with asymmetric clipping and token-level loss aggregation. Let $\mathcal{B}$ denote the combined training batch. Each training sample $(q_i, y_i, A_i) \in \mathcal{B}$ consists of a conditioning prefix q_i, a generated text continuation y_i (either a tree edge segment or a contrastive suffix), and the normalized advantage A_i. The objective is:

$$\mathcal{J}_{\text{DAPO}}(\theta) = \frac{1}{\sum_{i\in\mathcal{B}}|y_i|}\sum_{i\in\mathcal{B}}\sum_{t=1}^{|y_i|}\min\left(\rho_{i,t}(\theta)A_i,\ \text{clip}\left(\rho_{i,t}(\theta),\,1-\varepsilon_{\text{low}},\,1+\varepsilon_{\text{high}}\right)A_i\right), \tag{32}$$

where θ denotes the current policy parameters, and the token-level importance weight is defined as:

$$\rho_{i,t}(\theta) = \frac{\pi_\theta(y_{i,t} \mid q_i, y_{i,<t})}{\pi_{\theta_{\text{old}}}(y_{i,t} \mid q_i, y_{i,<t})}, \tag{33}$$

with θ_{old} denoting the behavior-policy snapshot that generated the rollout batch.

We use asymmetric clipping with $(\varepsilon_{\text{low}}, \varepsilon_{\text{high}}) = (0.2, 0.28)$.

B EXPERIMENTAL SETTINGS AND ANALYSIS PROTOCOLS

B.1 EXPERIMENTAL DETAILS AND HYPERPARAMETERS

Table 7 summarizes the training and evaluation hyperparameters.

Training. Each rollout batch samples prompts without replacement and receives one policy update.

Table 7: Key training and evaluation hyperparameters across model backbones.

Parameter	Value
Rollout batch size	128 prompts
Training sampling	Temperature 1.0, top-$p = 1.0$
Evaluation sampling	Temperature 0.6, top-$p = 0.95$
Optimizer	Adam
Learning rate	10^{-6}
Adam betas (β_1, β_2)	$(0.9, 0.999)$
Weight decay	0.1
Gradient clipping	1.0
DAPO clipping $(\varepsilon_{\text{low}}, \varepsilon_{\text{high}})$	$(0.2, 0.28)$
Maximum tree depth D	5
Branching factor B	6
Leaf-path multiplicity M	6
Root prior variance σ^2_{root}	1000
LMMSE regularization λ	10^{-8}
Suffix-pair tree accuracy threshold	0.75
Suffix minimum gap $L_{\text{long}} - L_{\text{short}}$	≥ 200 tokens
Suffix length ratio $L_{\text{long}}/L_{\text{short}}$	≥ 2.0
Suffix batch proportion cap ρ_{suf}	≤ 0.10

B.2 POST-MATCH CONTINUATION MEASUREMENT

Following Liu & Wang (2025), we truncate each completed reasoning trajectory on MATH500 at fractions $p \in \{0.1, 0.2, 0.3, \ldots, 0.9\}$ of its total generated token count. At each cutoff, we query the model with the following suffix:

```
Based on the reasoning above, state the final answer within
\boxed{}.
```

We identify the earliest truncation fraction $p^\star$ where the extracted candidate answer matches the final answer of the full trajectory. The post-match continuation count is $(1 - p^\star)L_{\text{total}}$, where L_{total} is the full trajectory length in tokens. We average these counts across trajectories for each problem and method.

B.3 GPT-5.2 ANNOTATION PROMPTS

Each segment is annotated for critical error, reflection, and successful reflection using three GPT-5.2 (OpenAI, 2025a) prompts. Their shared input template, JSON output schema, and individual instructions are collected below.

```
SHARED INPUT
Problem: {problem}
Ground-truth final answer: {answer}
Complete solution trajectory: {full_solution}
Segment to judge: {segment}

JSON OUTPUT FORMAT
{
  "label": "yes" | "no",
  "justification": "<one sentence citing the specific text>"
}

1. CRITICAL ERROR
Judge only the segment above: does this segment introduce a new
logical or computational error that turns an otherwise-correct line of
reasoning toward a wrong result? Errors inherited from earlier
segments do not count.

2. REFLECTION
Judge only the segment above: does this segment re-examine, question,
or verify earlier reasoning --- for example by revisiting or
re-deriving a previous step, checking an intermediate result, or
testing a candidate answer against the problem's conditions? Plain
forward continuation of the reasoning counts as no.

3. SUCCESSFUL REFLECTION (asked only when reflection = yes)
(a) Before this segment, the trajectory was heading toward an
incorrect final answer; (b) the reflection here redirects it to
ultimately reach the ground-truth answer. Return yes only if both
hold.
```

C ADDITIONAL ANALYSES AND ABLATIONS

C.1 QUALITATIVE REFLECTION EXAMPLE ON AIME 2024

Figure 6 illustrates the segment annotations and final outcomes of 16 sampled trajectories per method on one AIME24 problem.

(a) **DAPO** — 5 / 16 correct

#	S0	S1	S2	S3	S4	S5	S6	S7	Ans.
1	CE	CE	FR	FR	CE+FR	FR	FR	CE+FR	Y
2	–	–	–	–	–	–	–	–	Y
3	CE	–	CE	CE	FR	–	CE+FR	SR	Y
4	–	–	–	–	–	–	–	–	Y
5	–	–	–	–	–	–	–	–	Y
6	CE	–	FR	FR	FR	CE+FR	CE+FR	CE+FR	N
7	–	–	–	–	CE	FR	FR	CE+FR	N
8	–	CE	CE+FR	–	CE	FR	FR	CE+FR	N
9	CE	–	–	–	–	–	–	–	N
10	–	–	–	FR	FR	CE	FR	CE	N
11	–	–	CE	FR	FR	–	FR	CE+FR	N
12	–	–	–	CE	–	–	–	–	N
13	–	–	CE	CE	FR	–	CE	CE+FR	N
14	CE	CE	–	–	FR	FR	–	CE+FR	N
15	–	–	–	FR	FR	FR	FR	FR	N
16	CE	FR	–	FR	–	–	–	FR	N

(b) **CAST** — 10 / 16 correct

#	S0	S1	S2	S3	S4	S5	S6	S7	Ans.
1	–	CE	CE	–	CE	FR	SR	–	Y
2	–	–	–	–	–	–	–	–	Y
3	–	–	–	–	–	CE	SR	–	Y
4	–	–	–	–	CE+SR	–	–	–	Y
5	–	–	–	–	–	CE	–	–	Y
6	–	–	–	–	FR	FR	SR	–	Y
7	–	–	–	–	–	–	–	CE+SR	Y
8	–	–	–	–	–	–	–	–	Y
9	–	–	CE	SR	–	–	–	–	Y
10	–	–	–	–	–	–	–	–	Y
11	–	CE	CE+FR	–	CE	CE	CE+FR	CE+FR	N
12	–	–	–	SR	–	–	–	–	N
13	–	–	–	–	CE+FR	SR	–	–	N
14	CE+FR	CE	FR	CE+FR	CE+FR	FR	CE+FR	CE+FR	N
15	–	CE	CE	FR	FR	FR	FR	CE+FR	N
16	–	CE	CE	–	CE+FR	FR	CE+FR	CE+FR	N

CE: critical error SR: successful reflection FR: failed reflection

Figure 6: Segment labels for 16 trajectories per method on one AIME24 problem. Each row contains eight equal-length segments (S_0–S_7). CE: critical error; FR: failed reflection; SR: successful reflection. Compound labels use the reflection-outcome color. The final column gives verified answer correctness (Y/N).

In this example, CAST produces more correct responses than DAPO. Several CAST trajectories recover from earlier errors through reflection and reach the correct answer.

C.2 Ablation on suffix comparison

Both variants in Table 8 use the same tree schedule ($D = 5, B = 6, M = 6$) and topology-aware edge values. The ablated variant uses only $\mathcal{B}_{\text{edge}}$; full CAST adds suffix samples with the filters in Eq. 30.

Table 8: Benchmark breakdown evaluating the effect of suffix comparison on Qwen3-4B-Base. Acc. denotes percentage accuracy (%) and Avg. len denotes mean generated token count.

Setting	AIME24		AIME25		MATH500		Average	
	Acc.	Avg. len	Acc.	Avg. len	Acc.	Avg. len	Acc.	Avg. len
w/o suffix comparison	29.38	4081	**26.88**	3712	87.00	2001	**47.75**	3265
CAST	**30.10**	**3300**	25.42	**2856**	**87.51**	**1383**	47.68	**2513**

Suffix comparison reduces mean response length from 3265 to 2513 tokens (23.0%), while average accuracy changes from 47.75% to 47.68%. Responses are shorter on all three benchmarks. Accuracy increases on AIME24 and MATH500, but decreases from 26.88% to 25.42% on AIME25.

D Limitations and Future Work

Our evaluation covers mathematical reasoning with verifiable final answers; transfer to code generation, formal proving, or tasks with learned verifiers remains untested. The fixed tree budgets also impose a trade-off between rollout compute and branch resolution accuracy, as finite sampling may occasionally classify partially mixed prefixes as resolved. Adaptive allocation of depth, branching, and leaf samples is a natural direction for future work. Introducing a learned value model remains a direction for future research.