MCTS IS COUNTING THE WRONG THING: LEARNING EXPLORATION COUNTS FOR LLM REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Monte Carlo tree search (MCTS) has powered landmark successes in reinforcement learning, yet its gains in LLM reasoning can remain modest despite substantial search effort. We identify an overlooked bottleneck: *what MCTS counts*. Standard UCT assigns independent visit counts to textual branches, allowing different formulations of the same reasoning strategy to repeatedly receive exploration bonuses. Our theoretical analysis shows that this *count fragmentation* can impair exploration efficiency, as quantified by increased cumulative regret. To address this problem, we introduce Learned Pseudo-Count MCTS (LPC-MCTS), which changes only the exploration statistic in UCT. A lightweight Coin-Flipping Network head on the policy's hidden representations estimates representation coverage, enabling related branches to share exploration evidence. Across 5 mathematical-reasoning and 7 search benchmarks, LPC-MCTS consistently outperforms strong MCTS baselines. Holding each LPC-MCTS-trained policy fixed, replacing standard MCTS counts at inference improves average mean@16 accuracy across 5 mathematical benchmarks by 3.79 percentage points on Qwen2.5-7B and 2.71 points on Qwen3-4B. Applying LPC-MCTS during both training and inference yields gains of 4.96 and 4.37 points over standard MCTS, respectively. Together, our theory and experiments show that changing what MCTS counts can substantially improve its exploration efficiency and reasoning performance.

1 INTRODUCTION

Monte Carlo tree search (MCTS) is a powerful approach to sequential decision making (Kocsis & Szepesvári, 2006; Browne et al., 2012). By combining search with learned policies and value estimates, it helped enable landmark reinforcement learning systems such as AlphaGo (Silver et al., 2016). Its appeal extends naturally to large language models (LLMs): reasoning requires choosing among alternative plans, intermediate hypotheses, and tool calls, and search offers a way to revisit these choices before committing to an answer. A growing body of work has applied MCTS to LLM reasoning at inference time and during training (Hao et al., 2023; Zhou et al., 2023; Qi et al., 2025; Xie et al., 2024; Zhang et al., 2024; Wu et al., 2026). Meanwhile, methods such as Group Relative Policy Optimization (GRPO) obtain strong reasoning policies from groups of sampled responses without explicitly constructing search trees (Shao et al., 2024). For MCTS to offer a compelling alternative, its additional computation must translate into sufficiently better reasoning.

However, tree-search-based training can still yield only modest performance gains over GRPO-style methods (Wu et al., 2026). Zhao et al. (2026) report that tree-based approaches such as TreePO (Li et al., 2025b) perform comparably to GRPO and its variants. The DeepSeek-R1 report also highlights the difficulty of iteratively improving model performance through MCTS-based self-search (Guo et al., 2025). Together, these findings indicate that using a tree structure alone does not guarantee substantial gains in LLM reasoning. The effectiveness of exploration within the tree therefore remains an important question. With this in mind, our starting point is a component that is usually treated as routine bookkeeping: the visit count. We identify a *mismatch* between what MCTS counts and what language reasoning needs to explore. Our central finding is strikingly simple: *changing only the exploration count is enough to produce substantial gains.*

The issue arises from the meaning of a visit in a language search tree. UCT allocates exploration according to branch values and visit counts, giving larger exploration bonuses to less-visited branches

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.

(Kocsis & Szepesvári, 2006). When branches are identified by token sequences, however, different formulations of the same strategy receive independent counters. Repeatedly exploring one formulation does not increase the count of another. The counts therefore record textual branch visits accurately while failing to reflect how extensively a reasoning strategy has already been explored. We refer to this mismatch as *count fragmentation*. It allows familiar strategies to repeatedly appear novel and compete for exploration budget through different textual realizations.

Figure 1 illustrates this issue with an actual rollout-tree example. Branches *A* and *B* follow the same substitution-based derivation and reach the same quadratic, despite their different textual forms, whereas branch *C* instead uses a product identity and symmetric sums. In standard UCT, although *A* has already been explored 8 times, standard UCT treats the new branches *B* and *C* as equally unexplored because both have one exact visit. LPC-MCTS instead assigns similar pseudo-counts to *A* and *B* while separating *C*, allowing the search to share evidence across the two related branches and prioritize the distinct reasoning pattern.

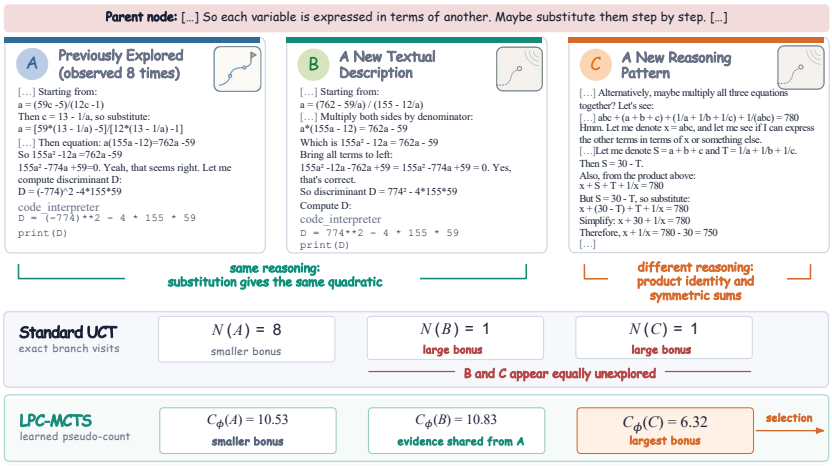


Figure 1: Motivation for LPC-MCTS. The parent and branches *A*-*C* are taken from an actual model rollout tree, and the displayed C_ϕ values are their learned pseudo-counts. Although branches *A* and *B* differ textually, they follow the same reasoning pattern and receive similar pseudo-counts, whereas *C* follows a distinct pattern. Standard UCT instead treats *B* and *C* as equally unexplored because both have one exact visit. Additional rollout-tree examples are provided in Appendix F.1.2.

We address this problem with *Learned Pseudo-Count MCTS* (LPC-MCTS), which retains the UCT selection structure and replaces independent branch-local visit counts with learned pseudo-counts. These counts estimate representation coverage, allowing exploration evidence to be shared across related branches. We obtain them using a Coin-Flipping Network (CFN) (Lobel et al., 2023), implemented as a lightweight auxiliary head on top of the policy's existing hidden representations. *The CFN head adds almost no computational overhead to standard MCTS.* Evidence collected from one branch can thereby increase the estimated familiarity of related branches, reducing repeated exploration of already-covered reasoning patterns and directing search toward less-explored alternatives.

We evaluate LPC-MCTS on two agentic settings: tool-integrated mathematical reasoning and search. The results show that it improves task performance over the strong MCTS baseline, indicating that representation-level coverage provides a more informative basis for allocating search than independent counts over textual branches. Our contributions are as follows:

- We introduce LPC-MCTS, a novel tree-search method that replaces independent branch counts with learned decision-level pseudo-counts, enabling search evidence to generalize across related LLM reasoning patterns, thereby reducing redundant exploration.
- We theoretically characterize the exploration cost of count fragmentation through cumulative regret.
- Our experiments on tool-integrated mathematical reasoning and search tasks demonstrate that LPC-MCTS improves task performance over strong MCTS baselines.

2 PRELIMINARIES

2.1 MCTS AND THE ROLE OF COUNTS

MCTS repeatedly allocates computation through four stages (Browne et al., 2012): (1) *selection* follows promising edges from the root; (2) *expansion* adds an unexplored child; (3) *simulation* estimates its outcome; and (4) *backpropagation* updates statistics along the selected path. Repeating these stages concentrates simulations on strong branches while reserving some budget for alternatives whose value remains uncertain.

Let node v represent state s_v, and let edge (v, a, v_a) represent taking action a and reaching child v_a. In tool-augmented LLM reasoning, the state is the interaction history; an action may be a reasoning segment, a tool call, or a final answer, and a tool may append an external observation. Let $Q(v, a)$ be the backed-up value of action a, $N(v)$ the number of simulations through v, and $N(v_a)$ the number allocated to its child branch. The UCT algorithm (Kocsis & Szepesvári, 2006) selects

$$a^* = \arg\max_{a \in \mathcal{A}(v)} \left[Q(v, a) + c\sqrt{\frac{\log(N(v) + 1)}{N(v_a) + \epsilon}} \right], \tag{1}$$

where $c > 0$ controls exploration and $\epsilon > 0$ ensures numerical stability. The first term favors high estimated value; the second is an optimistic bonus that is larger for less-visited children.

2.2 COIN-FLIPPING NETWORK

Exact visitation counts become uninformative in high-dimensional continuous spaces because nearly every state representation may be unique. Classical pseudo-count methods address this problem with learned density models (Bellemare et al., 2016; Ostrovski et al., 2017), but training and continually updating these models can be costly. A Coin-Flipping Network (CFN) provides a lighter alternative by reducing count estimation to random-label regression (Lobel et al., 2023). Whenever a representation $h_i \in \mathbb{R}^H$ is observed, the CFN pairs it with a randomly sampled Rademacher vector $z_i \in \{-1, +1\}^d$, whose coordinates independently take either value with probability $1/2$. Importantly, repeated occurrences receive different random targets. A predictor $f_\phi : \mathbb{R}^H \to \mathbb{R}^d$ is trained by minimizing the following objective:

$$\mathcal{L}_{\text{CFN}}(\phi) = \mathbb{E}_{(h_i, z_i) \sim \mathcal{D}_{\text{CFN}}} \left[\|f_\phi(h_i) - z_i\|_2^2 \right], \tag{2}$$

where $\mathcal{D}_{\text{CFN}}$ contains the observed representation-target pairs. If the representation h occurs n times, mean-squared loss minimization predicts the mean of its targets, $f^*(h) = \frac{1}{n}\sum_{i=1}^{n} z_i$. Since each target coordinate is independent, zero-mean, and unit-variance,

$$\mathbb{E}\left[\frac{1}{d} \|f^*(h)\|_2^2 \right] = \frac{1}{n}. \tag{3}$$

Thus, $\frac{1}{d}\|f_\phi(h)\|_2^2$ estimates an *inverse count*: it shrinks as evidence accumulates. The d coordinates estimate the same inverse count independently in this idealized solution, so averaging them reduces estimator variance. Taking its square root gives the CFN count-based bonus $b_\phi(h) = \sqrt{\frac{1}{d}\|f_\phi(h)\|_2^2}$, which scales as $\frac{1}{\sqrt{n}}$ under repeated observations.

3 METHODOLOGY

LPC-MCTS replaces the visit count with a learned pseudo-count that measures familiarity across model decisions. As shown in Figure 2, a CFN estimator computes this count from hidden representations, and the count directly determines the exploration bonus used for MCTS selection.

3.1 LEARNED DECISION-LEVEL PSEUDO-COUNTS

CFN estimation and training. We attach a lightweight linear CFN head to the policy's hidden representations. For $h \in \mathbb{R}^H$, we compute

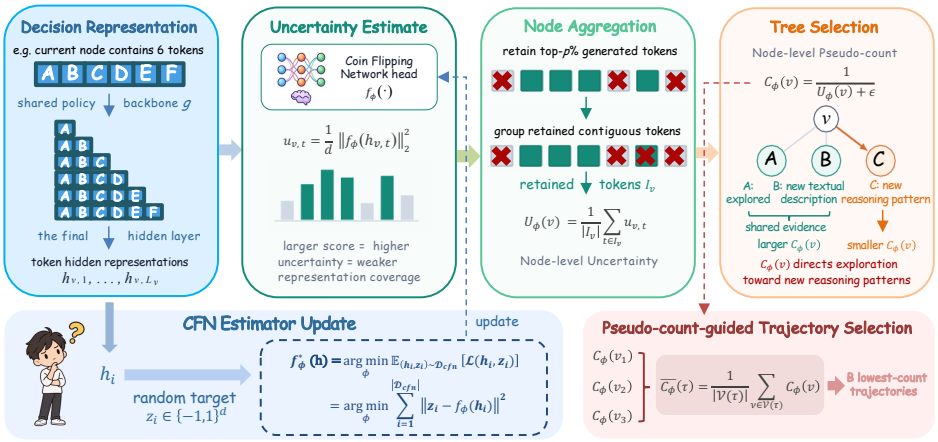


Figure 2: Overview of LPC-MCTS, including CFN-based pseudo-count estimation, estimator training, tree selection, and pseudo-count-guided trajectory selection.

$$f_\phi(h) = W_\phi^\top \operatorname{sg}(h), \qquad u_\phi(h) = \frac{1}{d}\|f_\phi(h)\|_2^2, \tag{4}$$

where $W_\phi \in \mathbb{R}^{H\times d}$ is trainable and $\operatorname{sg}(\cdot)$ prevents the CFN loss from updating the policy backbone. Following Lobel et al. (2023), we use $u_\phi(h)$ as a token-level inverse-count signal. During search, we collect representations at model-generated reasoning and tool-call positions, excluding prompt and tool-response tokens. Each observed occurrence h_i is paired with a fresh, independently sampled Rademacher target $z_i \in \{-1,+1\}^d$. We train the head on minibatches $\mathcal{B}_{\mathrm{CFN}}$ of the accumulated pairs using

$$\mathcal{L}_{\mathrm{CFN}}(\phi) = \frac{1}{|\mathcal{B}_{\mathrm{CFN}}|} \sum_{(h_i,z_i)\in\mathcal{B}_{\mathrm{CFN}}} \|f_\phi(h_i) - z_i\|_2^2 . \tag{5}$$

From token scores to a node count. For a search node v, let $x_v = (x_{v,1}, \ldots, x_{v,L_v})$ denote its token sequence. We encode this sequence independently with the policy backbone g and score each position with the trained CFN head f_ϕ:

$$h_{v,t} = g(x_{v,\le t}), \qquad u_{v,t} = \frac{1}{d}\|f_\phi(h_{v,t})\|_2^2. \tag{6}$$

By the definition above, $u_{v,t}$ is the representation-level uncertainty at position t. A larger score means that the CFN has accumulated less evidence for the represented state, making that position more important for exploration. Among the model-generated tokens in node v, we retain the top-$p\%$ with the largest $u_{v,t}$, so that the node statistic emphasizes the exploration signal from positions with the relatively higher uncertainty. Since isolated high scores may be noisy, we partition the retained positions into contiguous runs and discard short runs that do not form coherent local segments. Let I_v be the union of the remaining runs. We then define

$$U_\phi(v) = \frac{1}{|I_v|}\sum_{t\in I_v} u_{v,t}, \qquad C_\phi(v) = \frac{1}{U_\phi(v)+\epsilon}. \tag{7}$$

$U_\phi(v)$ summarizes uncertainty over coherent high-score regions. $C_\phi(v)$ is the reciprocal of $U_\phi(v)+\epsilon$. We use it as a learned pseudo-count: a larger uncertainty score produces a smaller $C_\phi(v)$ and thus, all else being equal, a larger exploration bonus in Eq. 8.

3.2 Pseudo-count-guided tree search

Constructing tree nodes. During expansion, we grow the search tree together with the generated trajectory and use token entropy as the signal for creating new nodes. After each tool response, the policy generates a probe window of w tokens. Let $\overline{H}_{\mathrm{win}}$ and $\overline{H}_{\mathrm{ctx}}$ denote the mean token entropy over the probe window and the preceding context, respectively. We create a new node when $\overline{H}_{\mathrm{win}} > \alpha\overline{H}_{\mathrm{ctx}}$; otherwise, generation continues within the current node. Here, $\alpha > 1$ controls the frequency of node branching, with a larger α leading to less frequent branching.

Shared rollout and tree values. During expansion, the behavior policy $\pi_{\theta_{\text{old}}}$ generates candidate action segments. Repeated selection and expansion yield terminal trajectories. A verifier or reward model assigns each completed trajectory τ an outcome reward $R(\tau)$, and these outcomes are used to maintain an action value $Q(v, a)$ for each expanded edge (v, a).

UCT-style acquisition rule. At a node v with expanded children, let $\mathcal{A}_{\text{active}}(v)$ denote their actions and let v_a be the child reached by action a. We place the parent normalizer on the same learned scale as the child pseudo-counts by setting $M_\phi(v) = \sum_{a \in \mathcal{A}_{\text{active}}(v)} C_\phi(v_a)$. LPC-MCTS selects

$$a^* = \arg \max_{a \in \mathcal{A}_{\text{active}}(v)} \left[Q(v, a) + c_{\text{count}} \sqrt{\frac{\log(M_\phi(v) + 1)}{C_\phi(v_a)}} \right]. \tag{8}$$

By construction, at a fixed selection step, a larger node score $U_\phi(v_a)$ gives a smaller $C_\phi(v_a)$ and hence a larger exploration bonus.

Policy optimization from searched trajectories. Following the node-value formulation of DeepSearch (Wu et al., 2026) and group-relative reward normalization (Shao et al., 2024; Chu et al., 2026), we compute an action-level advantage from the corresponding tree value. Let $\mathcal{L}_x$ denote the terminal trajectories for prompt x:

$$b_x = \frac{1}{|\mathcal{L}_x|} \sum_{\tau \in \mathcal{L}_x} R(\tau), \qquad A(v, a) = Q(v, a) - b_x. \tag{9}$$

Here $Q(v, a)$ denotes the tree value associated with edge (v, a), and b_x is the prompt-level mean terminal outcome used as the baseline. All generated tokens in an action segment $a = (y_1, \ldots, y_{L_a})$ share this action-level advantage.

We optimize these action segments using the clipped surrogate objective of PPO (Schulman et al., 2017). For token t, the importance ratio is $r_{\theta,t}(v, a) = \frac{\pi_\theta(y_t | p_v, y_{<t})}{\pi_{\theta_{\text{old}}}(y_t | p_v, y_{<t})}$, where p_v is the complete prefix at node v. Following DAPO (Yu et al., 2025), we use separate lower and upper clipping ranges and average the surrogate over all trainable tokens:

$$\mathcal{J}_{\text{policy}}(\theta) = \frac{1}{\sum_{(v,a) \in \mathcal{D}_{\text{tree}}} L_a} \sum_{(v,a) \in \mathcal{D}_{\text{tree}}} \sum_{t=1}^{L_a} \min \Big(r_{\theta,t}(v, a)\, A(v, a), \\ \text{clip}(r_{\theta,t}(v, a), 1 - \epsilon_{\text{low}}, 1 + \epsilon_{\text{high}})\, A(v, a) \Big), \tag{10}$$

We additionally use dual clipping for negative advantages to limit extreme updates from large importance ratios, as detailed in Appendix A.2. LPC-MCTS influences policy learning through exploration: the pseudo-count changes which branches are searched and hence which action segments enter $\mathcal{D}_{\text{tree}}$, while empirical visits remain in use for value averaging.

3.3 Pseudo-count-guided trajectory selection

Let $\mathcal{C}_x = \{\tau_1, \ldots, \tau_K\}$ be the candidate pool of completed trajectories for prompt x, and let $\mathcal{V}(\tau)$ denote the set of nodes on trajectory τ. For trajectory comparison, node scores within a candidate pool are evaluated against a shared reference state of f_ϕ. We define the pseudo-count of a trajectory as the mean of its node-level pseudo-counts:

$$\overline{C}_\phi(\tau) = \frac{1}{|\mathcal{V}(\tau)|} \sum_{v \in \mathcal{V}(\tau)} C_\phi(v), \qquad \mathcal{B}_x = \arg \min_{\substack{\mathcal{B} \subseteq \mathcal{C}_x \\ |\mathcal{B}| = B}} \sum_{\tau \in \mathcal{B}} \overline{C}_\phi(\tau). \tag{11}$$

We sort $\mathcal{C}_x$ by $\overline{C}_\phi(\tau)$ in ascending order and pass the first B trajectories, denoted by $\mathcal{B}_x$, to policy optimization. This rule prioritizes trajectories that pass through underexplored decisions. It thereby extends pseudo-count-guided exploration from tree expansion to training-batch construction.

4 How Count Fragmentation Impairs Exploration

We next analyze how count fragmentation affects exploration efficiency. We quantify this effect through *cumulative regret*, a standard criterion in online learning that measures the reward lost relative to always selecting an optimal action (Auer et al., 2002; Lattimore & Szepesvári, 2020). We

begin with the simplest setting, a one-step decision problem modeled as a stochastic bandit, and show that count fragmentation already incurs an additional exploration cost.

Consider a fixed finite bandit with $K \geq 2$ groups of actions. Within each group, actions are distinct choices but yield rewards from the same distribution. Let $\mathcal{C}_c$ denote group c, containing $m_c = |\mathcal{C}_c|$ actions. Rewards lie in $[0, 1]$, are independent across pulls, and have mean μ_c for every action in group c. Assume group 1 contains the unique optimal action, so $m_1 = 1$ and $\mu_1 > \mu_c$ for all $c \geq 2$. Let $\Delta_c = \mu_1 - \mu_c$ denote the reward gap of a suboptimal group. For actions $A_1, \ldots, A_T$ selected over T rounds, the expected cumulative regret is $R(T) = \mathbb{E}\Big[\sum_{t=1}^{T}\big(\mu_1 - \mu_{c(A_t)}\big)\Big] = \sum_{c=2}^{K} \Delta_c \, \mathbb{E}[N_c(T)]$, where $c(a)$ identifies the group containing action a, and $N_c(T)$ counts all selections of actions in group c. Regret therefore measures the sampling budget spent on suboptimal actions, weighted by their reward gaps.

Independent versus shared counts. Let $N_a(t)$ and $\widehat{\mu}_a(t)$ denote the visit count and empirical mean of action a after t rounds, and define the class count as $N_c(t) = \sum_{a \in \mathcal{C}_c} N_a(t)$. We compare two UCB rules that differ only in their exploration counts. Standard UCB1 uses $I_{\mathrm{ind}}(a, t) = \widehat{\mu}_a(t-1) + \sqrt{2 \log t / N_a(t-1)}$, whereas the shared-count variant uses $I_{\mathrm{share}}(a, t) = \widehat{\mu}_a(t-1) + \sqrt{2 \log t / N_{c(a)}(t-1)}$. Both sample every action once before selecting actions with the largest index. Each action retains its own empirical mean, computed only from its observed rewards; only the exploration counts are shared.

Theorem 1 (Regret penalty of count fragmentation)**.** *For the bandit and selection rules defined above, as $T \to \infty$,*

$$R_{\mathrm{ind}}(T) = 2 \log T \sum_{c=2}^{K} \frac{m_c}{\Delta_c} + o(\log T), \qquad R_{\mathrm{share}}(T) = 2 \log T \sum_{c=2}^{K} \frac{1}{\Delta_c} + o(\log T). \tag{12}$$

In particular, if every suboptimal class contains m equivalent actions, then $R_{\mathrm{ind}}(T)/R_{\mathrm{share}}(T) \to m$.

Independent counts therefore repeat the same logarithmic exploration cost across equivalent actions, increasing regret even when rewards are deterministic. The proof and a finite-time bound are provided in Appendix B.

Extension to tree search. We extend our analysis to recursive UCT with terminal-reward backup and obtain a similar regret separation. In the tree setting, the leading exploration costs accumulate along the optimal path, with independent counts paying separately for each equivalent action. When every suboptimal class on this path contains m equivalent actions, independent-count UCT incurs m times the leading cumulative regret of shared-count UCT, while both retain separate action-value estimates. The formal statement (Theorem 2), assumptions, and proof are provided in Appendix C.

5 Related Work

Tree-structured search for LLM agents. Tree of Thoughts and RAP formulate language reasoning as explicit search over intermediate states (Yao et al., 2023a; Hao et al., 2023), while ReAct and LATS extend this view to agents that interleave reasoning, actions, and environmental observations (Yao et al., 2023b; Zhou et al., 2023). Subsequent work augments tree search with tool-aware feedback, reflective simulation, safe pruning, adaptive lookahead, and multi-agent reasoning (Yang et al., 2026; Zhang et al., 2026c; Qian et al., 2026; Zhang et al., 2026a; Li et al., 2026). Recent training-time methods use the tree structure to obtain backed-up node values or fine-grained relative advantages, improving rollout efficiency and credit assignment (Wu et al., 2026; Ji et al., 2026; Li et al., 2025b). LPC-MCTS addresses a complementary question: how exploration evidence should be represented during tree selection. Some methods reduce redundant exploration through embedding-based clustering and merging of semantically similar search states. For instance, FETCH uses a separate embedding model to form these clusters (Wang et al., 2025). Compared with this class of explicit clustering methods, LPC-MCTS requires neither an additional embedding model nor online clustering. Its lightweight CFN head operates on the policy's existing hidden representations and is updated with states collected during rollouts. The pseudo-count therefore reflects how familiar a state is under the current policy after accumulated rollout visitation and enters tree selection directly.

Count-based exploration beyond tabular spaces. UCT treats the visit count as the evidence supporting a branch (Kocsis & Szepesvári, 2006). Exact counts are effective when search repeatedly reaches the same discrete states, but language trees contain many low-visit nodes whose textual forms differ even when their reasoning content is similar. Generalized pseudo-counts address this problem in large observation spaces by replacing exact identity with learned density or representation-level recurrence (Bellemare et al., 2016; Ostrovski et al., 2017). Coin-Flipping Networks (CFNs) estimate such recurrence through random-label regression, avoiding an explicit normalized density model (Lobel et al., 2023). Recent work has brought these exploration signals to LLM training. COPO uses a CFN pseudo-count over prompt–response pairs for optimistic preference alignment (Bai et al., 2025), while MERCI applies token-level CFN scores as intrinsic rewards for diverse reasoning trajectories (Zhang et al., 2026b). CDE instead combines actor surprise with critic disagreement (Dai et al., 2026). These methods shape policy learning after responses are sampled, whereas LPC-MCTS uses the learned pseudo-counts directly during tree selection.

6 Experiments

To evaluate whether learned pseudo-counts improve exploration and final task performance, we study LPC-MCTS in two agentic settings: tool-integrated mathematical reasoning and search.

6.1 Experimental Setup

Tool-integrated mathematical reasoning. We use Qwen2.5-7B (Yang et al., 2025b) and Qwen3-4B (Yang et al., 2025a) as policy backbones. Both models are initialized with the same ReTool SFT dataset (Feng et al., 2026). For RL training, we use DAPO-Math-17K (Yu et al., 2025). We evaluate on AIME 2024/2025/2026 (MAA, 2025), as well as HMMT February 2025 and BRUMO 2025 from MathArena (Dekoninck et al., 2026). For each problem, we sample 16 responses with temperature $T = 0.6$ and top-$p = 0.95$, and report their mean accuracy (mean@16).

Search. We use Qwen2.5-7B-Instruct and Llama-3.2-3B-Instruct (Grattafiori et al., 2024) as policy backbones. Both models are initialized by SFT on the same 2,000 examples sampled from the dataset constructed by Li et al. (2025a). Following Jin et al. (2025), our RL training set consists of the training splits of NQ (Kwiatkowski et al., 2019) and HotpotQA (Yang et al., 2018). We evaluate our training paradigm on 7 search-augmented QA tasks, including single-hop QA datasets (NQ, TriviaQA (Joshi et al., 2017), and PopQA (Mallen et al., 2023)) and multi-hop QA datasets (HotpotQA, 2Wiki (Ho et al., 2020), MuSiQue (Trivedi et al., 2022), and Bamboogle (Press et al., 2023)). We use offline retrieval over the 2018 Wikipedia corpus (Karpukhin et al., 2020) using E5-base-v2 embeddings (Wang et al., 2024), and return the top-3 documents for each query. We evaluate each question with greedy sampling and MCTS. For MCTS, we set the temperature $T = 0.6$ and top-$p = 0.95$, and report the mean accuracy over 5 responses per question (mean@5).

Baselines and configurations. We use DAPO as our policy-optimization baseline and additionally include two exploration-oriented methods. Cheng et al. (2025) encourage exploration through entropy-based advantage shaping; this baseline is denoted as DAPO + Entropy. Gao et al. (2026) use random network distillation (RND)-based intrinsic rewards for exploration; this baseline is denoted as DAPO + IMAGINE. We compare these DAPO-based baselines with standard visit-count MCTS and LPC-MCTS, which jointly trains the policy and CFN head. During tree search, standard MCTS uses discrete visit counts, whereas LPC-MCTS replaces them with CFN-estimated pseudo-counts. For MCTS-based inference, motivated by the use of self-consistency-derived terminal rewards for MCTS backup in rStar (Qi et al., 2025), we assign terminal rewards through answer consistency without using reference answers. After adding each terminal node, we group all terminal nodes by their final answers, assign $+1$ to nodes in the current majority group and -1 to the others, and recompute the backed-up Q-values throughout the tree. Details are provided in Appendix E.

6.2 Analysis of the Learned CFN Head

Token-level uncertainty visualization. Within each selected response, we visualize how the CFN score $u_{v,t}$ of every eligible generated token changes across the sequence. The visualizations and corresponding analysis are provided in Appendix F.1.1. Across these examples, high scores cluster on

Table 1: Mathematical reasoning results under different training and test-time inference settings. The **bold text** represents the best, while underlined represents the second best.

Training	Inference	AIME 24	AIME 25	AIME 26	HMMT25	BRUMO25	Avg.
Qwen2.5-7B							
DAPO	Direct sampling	24.58	23.33	20.21	9.38	30.63	21.63
DAPO	MCTS	29.79	27.71	23.33	16.25	40.63	27.54
DAPO + Entropy	Direct sampling	27.08	22.92	23.96	12.29	33.30	23.91
DAPO + Entropy	MCTS	30.42	27.50	26.25	14.17	42.08	28.08
DAPO + IMAGINE	Direct sampling	24.38	21.46	20.21	10.83	32.08	21.79
DAPO + IMAGINE	MCTS	28.33	26.46	23.96	16.04	41.88	27.33
MCTS	Direct sampling	25.21	23.75	21.04	11.25	30.42	22.33
MCTS	MCTS	28.33	27.50	26.04	15.21	42.08	27.83
LPC-MCTS	Direct sampling	26.67	24.38	22.08	13.33	34.79	24.25
LPC-MCTS	MCTS	30.83	29.17	25.42	17.08	42.50	29.00
LPC-MCTS	LPC-MCTS	**37.08**	**30.63**	**27.92**	**23.33**	**45.00**	**32.79**
Qwen3-4B							
DAPO	Direct sampling	59.58	51.46	54.17	36.30	58.30	51.96
DAPO	MCTS	71.88	57.50	59.38	47.92	61.67	59.67
DAPO + Entropy	Direct sampling	60.63	53.33	55.83	35.63	61.46	53.38
DAPO + Entropy	MCTS	69.17	56.04	64.17	43.96	**68.33**	60.33
DAPO + IMAGINE	Direct sampling	62.91	49.38	57.29	37.08	58.75	53.08
DAPO + IMAGINE	MCTS	69.58	55.42	61.46	49.38	65.42	60.25
MCTS	Direct sampling	60.83	52.29	57.50	35.83	60.00	53.29
MCTS	MCTS	73.75	56.46	65.00	50.21	65.21	62.13
LPC-MCTS	Direct sampling	62.71	53.13	58.13	36.67	59.38	54.00
LPC-MCTS	MCTS	71.25	59.17	**74.38**	50.63	63.54	63.79
LPC-MCTS	LPC-MCTS	**78.13**	**63.54**	72.92	**51.88**	66.04	**66.50**

novel reasoning patterns, such as introducing a new solution strategy, applying a theorem, correcting an earlier step, reformulating a query, or integrating evidence into an answer. In contrast, common connecting text and simple calculations or routine queries receive lower scores. The high values also extend across groups of nearby tokens rather than appearing as isolated spikes. Together, these observations suggest that the learned CFN head assigns higher uncertainty to novel high-level reasoning patterns in both mathematical reasoning and search, rather than isolated token differences. This behavior directly supports our learned pseudo-count: Eq. 7 maps less-familiar reasoning spans with larger $U_\phi(v)$ to a smaller $C_\phi(v)$ and thus a larger exploration bonus in Eq. 8, while familiar patterns receive larger learned counts and less exploration.

Reasoning-pattern sensitivity. To test whether the learned pseudo-count captures reasoning-pattern similarity during search, we sample some sibling groups from actual model rollout trees. In each group, two textually different children follow the same or a closely related reasoning pattern, while a third child takes a distinct approach. The two pattern-aligned children receive closely matched pseudo-counts, whereas the distinct child is separated from both by a markedly larger gap. This separation indicates that the estimator responds to changes in reasoning pattern rather than surface-level wording alone. In these cases, the distinct child also receives a lower pseudo-count and therefore a larger exploration bonus in Eq. 8. The complete cases are analyzed in Appendix F.1.2. These results qualitatively support the motivation in Figure 1: discouraging redundant expansion of familiar patterns while preserving exploration for distinct strategies.

6.3 Main Results

Mathematical reasoning. Table 1 shows that LPC-MCTS achieves the highest average performance on both backbones when used during both training and inference, improving over standard MCTS by 4.96 points on Qwen2.5-7B and 4.37 points on Qwen3-4B. Holding each LPC-MCTS-trained policy fixed, replacing discrete visit counts with learned pseudo-counts at inference yields further gains of 3.79 and 2.71 points, isolating the benefit of LPC-MCTS during test-time search.

Table 2: Search results under different training and test-time inference strategies. We evaluate greedy sampling with temperature $T = 0$ and MCTS with temperature $T = 0.6$.

Training	Inference	NQ	TriviaQA	PopQA	HotpotQA	2Wiki	MuSiQue	Bamboogle	Avg.
Qwen2.5-7B-Instruct									
DAPO	Greedy sampling	43.40	55.60	39.60	46.00	43.60	31.20	46.40	43.69
DAPO	MCTS	45.48	56.64	**42.48**	45.96	40.04	31.44	48.00	44.29
DAPO + Entropy	Greedy sampling	44.60	57.60	41.00	44.60	44.60	31.40	48.20	44.57
DAPO + Entropy	MCTS	46.08	56.56	40.68	46.08	40.08	32.00	48.96	44.35
DAPO + IMAGINE	Greedy sampling	44.40	56.60	40.60	44.20	42.40	32.20	50.40	44.40
DAPO + IMAGINE	MCTS	46.80	56.52	40.12	46.04	39.56	32.44	47.52	44.14
MCTS	Greedy sampling	42.40	58.00	41.60	46.20	43.00	31.60	46.40	44.17
MCTS	MCTS	45.44	58.24	40.80	46.28	42.88	31.16	47.20	44.57
LPC-MCTS	Greedy sampling	44.40	57.00	42.00	48.00	43.20	33.40	48.80	45.26
LPC-MCTS	MCTS	45.32	**58.56**	42.32	**48.20**	46.08	33.16	50.08	46.25
LPC-MCTS	LPC-MCTS	**46.92**	58.48	42.12	47.96	**46.88**	**34.24**	**51.20**	**46.83**
Llama-3.2-3B-Instruct									
DAPO	Greedy sampling	44.40	55.00	36.60	44.60	36.80	30.80	47.20	42.20
DAPO	MCTS	44.52	55.92	37.92	45.08	39.04	30.32	46.24	42.72
DAPO + Entropy	Greedy sampling	45.60	55.00	40.00	45.00	38.40	31.40	46.40	43.11
DAPO + Entropy	MCTS	46.52	55.80	40.80	45.68	37.68	**32.00**	45.92	43.49
DAPO + IMAGINE	Greedy sampling	44.00	55.20	40.80	44.80	40.00	30.40	47.20	43.20
DAPO + IMAGINE	MCTS	46.32	55.76	40.72	44.56	39.28	31.40	47.52	43.65
MCTS	Greedy sampling	45.20	55.40	38.80	44.00	**43.20**	28.60	48.00	43.31
MCTS	MCTS	45.28	56.04	39.80	44.56	40.28	30.04	48.32	43.47
LPC-MCTS	Greedy sampling	45.00	**56.80**	**43.00**	46.00	40.40	30.40	48.80	44.34
LPC-MCTS	MCTS	**47.16**	56.00	41.88	47.12	41.68	30.92	48.32	44.73
LPC-MCTS	LPC-MCTS	46.80	56.16	41.84	**47.40**	42.28	31.72	**50.08**	**45.18**

Under standard MCTS evaluation, LPC-MCTS-trained policies also outperform the strongest alternative training method by 0.92 and 1.66 points. The consistent improvements across both backbones and both stages provide strong empirical support that mitigating count fragmentation benefits test-time exploration and the collection of trajectories for policy optimization.

Search. Table 2 shows improvements on search. Using LPC-MCTS at both stages improves over standard MCTS by 2.26 points on Qwen2.5-7B-Instruct and 1.71 points on Llama-3.2-3B-Instruct. Holding the policy fixed, LPC-MCTS improves test-time MCTS by 0.58 and 0.45 points, while LPC-MCTS training improves MCTS evaluation by 1.68 and 1.08 points over the strongest alternatives. Compared with mathematical reasoning, however, these gains are more modest. The logged trees provide one explanation: among 256 sampled trees per task, search covers 1.82 effective first-level branches on average, compared with 4.05 for mathematical reasoning. This narrower structure suggests that search is simpler in our setting and tends to follow more standardized solution strategies, leaving fewer competing branches across which pseudo-counts can share exploration evidence.

The ablations in Appendix F.2.2 show that pseudo-count-guided trajectory selection and token-to-node aggregation both improve performance, while the sensitivity results in Appendix F.2.3 favor $p = 30$ on both backbones. Overall, LPC-MCTS improves performance in both domains *with almost no computational overhead from its CFN head*. The larger gains in mathematical reasoning are consistent with richer branching, where learned pseudo-counts can better distinguish novel paths and share exploration evidence across related branches.

7 Conclusion

We identify count fragmentation as a key limitation of MCTS for LLM reasoning: textually different branches can follow the same reasoning pattern while retaining independent visit counts. We introduce LPC-MCTS, which replaces exact UCT visits with pseudo-counts from a lightweight CFN head and prioritizes less-covered trajectories for policy optimization. Our theory quantifies the additional exploration cost caused by fragmented counts. Across mathematical reasoning and search tasks, LPC-MCTS consistently improves over standard MCTS, while its learned counts align across related patterns and distinguish novel ones, showing that representation-level familiarity provides a more effective basis for exploration than textual branch identity.

AI USE STATEMENT

In this work, we used generative AI tools to assist with writing proofs, proposing or refining hypotheses, and translation. We did not use generative AI tools to generate synthetic data sets, help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, design or provide feedback on research methodology or experiments, implement methods, clean and reformat datasets, support qualitative and thematic data analysis, or interpret results. Additionally, we used generative AI tools to create or modify scientific figures or images, create or edit software code, summarize or analyse existing literature, and edit a research paper to improve readability. We reviewed all AI-assisted material, checked the mathematical proofs and their assumptions, verified the cited references, validated figures against the underlying experimental records, and reviewed all AI-generated code for correctness. We take full responsibility for the accuracy and originality of the final work, including all content and code prepared with AI assistance.

ETHICS STATEMENT

This work adheres to the ICLR Code of Ethics. In this study, no human subjects or animal experimentation was involved. All datasets used were sourced in compliance with relevant usage guidelines, ensuring no violation of privacy. We have taken care to avoid any biases or discriminatory outcomes in our research process. No personally identifiable information was used, and no experiments were conducted that could raise privacy or security concerns. We are committed to maintaining transparency and integrity throughout the research process.

REPRODUCIBILITY STATEMENT

We have made every effort to ensure that the results presented in this paper are reproducible. All models and datasets used in our work are publicly available, and the code is openly available at: `https://anonymous.4open.science/r/lpc-mcts-D449`. The experimental setup, including training steps, model configurations, and hardware details, is described in detail in the appendix. We have also provided a full description to assist others in reproducing our experiments.

REFERENCES

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

Chenjia Bai, Yang Zhang, Shuang Qiu, Qiaosheng Zhang, Kang Xu, and Xuelong Li. Online preference alignment for language models via count-based exploration. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=cfKZ5VrhXt`.

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012. doi: 10.1109/TCIAIG.2012.2186810.

Daixuan Cheng, Shaohan Huang, Xuekai Zhu, Bo Dai, Wayne Xin Zhao, Zhenliang Zhang, and Furu Wei. Reasoning with exploration: An entropy perspective, 2025. URL `https://arxiv.org/abs/2506.14758`.

Xiangxiang Chu, Hailang Huang, Xiao Zhang, Fei Wei, and Yong Wang. GPG: A simple and strong reinforcement learning baseline for model reasoning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL `https://openreview.net/forum?id=inccdtfx8x`.

Pierre-Arnaud Coquelin and Rémi Munos. Bandit algorithms for tree search. *arXiv preprint cs/0703062*, 2007.

Runpeng Dai, Linfeng Song, Haolin Liu, Zhenwen Liang, Dian Yu, Haitao Mi, Zhaopeng Tu, Rui Liu, Tong Zheng, Hongtu Zhu, and Dong Yu. CDE: Curiosity-driven exploration for efficient reinforcement learning in large language models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=5rXN5knHKW.

Jasper Dekoninck, Nikola Jovanović, Tim Gehrunger, Kári Rögnvaldsson, Ivo Petrov, Chenhao Sun, and Martin Vechev. Beyond benchmarks: Matharena as an evaluation platform for mathematics with llms. 2026. URL https://arxiv.org/abs/2605.00674.

Lin Fan and Peter W Glynn. The typical behavior of bandit algorithms. *arXiv preprint arXiv:2210.05660*, 2022.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. Retool: Reinforcement learning for strategic tool use in LLMs. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=tRk1nofSmz.

Jingtong Gao, Ling Pan, Yejing Wang, Rui Zhong, Chi Lu, Maolin Wang, Qingpeng Cai, Peng Jiang, and Xiangyu Zhao. Navigate the unknown: Enhancing llm reasoning with intrinsic motivation guided exploration, 2026. URL https://arxiv.org/abs/2505.17621.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

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(8081):633–638, 2025.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 8154–8173, 2023.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps, 2020. URL https://arxiv.org/abs/2011.01060.

Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

Yuxiang Ji, Ziyu Ma, Yong Wang, Guanhua Chen, Xiangxiang Chu, and Liaoni Wu. Tree search for LLM agent reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=ZpQwAFhU13.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan O Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training LLMs to reason and leverage search engines with reinforcement learning. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=Rwhi91ideu.

Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension, 2017. URL https://arxiv.org/abs/1705.03551.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6769–6781, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.550. URL https://aclanthology.org/2020.emnlp-main.550/.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pp. 282–293. Springer, 2006.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019. doi: 10.1162/tacl_a_00276. URL https://aclanthology.org/Q19-1026/.

Tor Lattimore and Csaba Szepesvári. *Bandit Algorithms*. Cambridge University Press, 2020.

Weizhen Li, Jianbo Lin, Zhuosong Jiang, Jingyi Cao, Xinpeng Liu, Jiayu Zhang, Zhenqiang Huang, Qianben Chen, Weichen Sun, Qiexiang Wang, Hongxuan Lu, Tianrui Qin, Chenghao Zhu, Yi Yao, Shuying Fan, Xiaowan Li, Tiannan Wang, Pai Liu, King Zhu, He Zhu, Dingfeng Shi, Piaohong Wang, Yeyi Guan, Xiangru Tang, Minghao Liu, Yuchen Eleanor Jiang, Jian Yang, Jiaheng Liu, Ge Zhang, and Wangchunshu Zhou. Chain-of-agents: End-to-end agent foundation models via multi-agent distillation and agentic rl, 2025a. URL https://arxiv.org/abs/2508.13167.

Xiang Li, Yucheng Zhou, Xiangzhi Wei, Zesheng Shi, Haiyuan Wan, Gong Yifan, Fangming Liu, and Jing Li. Llm inductive reasoning through multi-agent enhanced monte carlo tree search. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 23548–23562, 2026.

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, 2025b. URL https://arxiv.org/abs/2508.17445.

Sam Lobel, Akhil Bagaria, and George Konidaris. Flipping coins to estimate pseudocounts for exploration in reinforcement learning. In *International Conference on Machine Learning*, pp. 22594–22613. PMLR, 2023.

MAA. American invitational mathematics examination (aime), 2025. https://maa.org/.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories, 2023. URL https://arxiv.org/abs/2212.10511.

Georg Ostrovski, Marc G Bellemare, Aäron Oord, and Rémi Munos. Count-based exploration with neural density models. In *International conference on machine learning*, pp. 2721–2730. PMLR, 2017.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models, 2023. URL https://arxiv.org/abs/2210.03350.

Zhenting Qi, Mingyuan Ma, Jiahang Xu, Li Lyna Zhang, Fan Yang, and Mao Yang. Mutual reasoning makes smaller LLMs stronger problem-solvers. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=6aHUmotXaw.

Tianhao Qian, Jiayu Chen, and Lixu Wang. Pac-cf: Calibrating irreversible frontier pruning in llm-guided search, 2026. URL https://arxiv.org/abs/2604.14345.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 2022.

Ante Wang, Linfeng Song, Ye Tian, Dian Yu, Haitao Mi, Xiangyu Duan, Zhaopeng Tu, Jinsong Su, and Dong Yu. Don't get lost in the trees: Streamlining llm reasoning by overcoming tree search exploration pitfalls. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23946–23959, 2025.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. Text embeddings by weakly-supervised contrastive pre-training, 2024. URL https://arxiv.org/abs/2212.03533.

Fang Wu, Weihao Xuan, Heli Qi, Aaron Tu, Ximing Lu, Li Erran Li, and Yejin Choi. Deepsearch: Overcome the bottleneck of reinforcement learning with verifiable rewards via tree-based search. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=Kx0G6v2c2S.

Yuxi Xie, Anirudh Goyal, Wenyue Zheng, Min-Yen Kan, Timothy P Lillicrap, Kenji Kawaguchi, and Michael Shieh. Monte carlo tree search boosts reasoning via iterative preference learning. *arXiv preprint arXiv:2405.00451*, 2024.

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

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025b. URL https://arxiv.org/abs/2412.15115.

Shuo Yang, Caren Han, Yihao Ding, Shuhe Wang, and Eduard Hovy. Tooltree: Efficient LLM tool planning via dual-feedback monte carlo tree search and bidirectional pruning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=Ef5O9gNNLE.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering, 2018. URL https://arxiv.org/abs/1809.09600.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=WE_vluYUL-X.

Deheng Ye, Zhao Liu, Mingfei Sun, Bei Shi, Peilin Zhao, Hao Wu, Hongsheng Yu, Shaojie Yang, Xipeng Wu, Qingwei Guo, Qiaobo Chen, Yinyuting Yin, Hao Zhang, Tengfei Shi, Liang Wang, Qiang Fu, Wei Yang, and Lanxiao Huang. Mastering complex control in moba games with deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 6672–6679, 2020.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, YuYue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, LingJun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Ru 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, Yonghui Wu, and Mingxuan Wang. DAPO: An open-source LLM reinforcement learning system at scale. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=2a36EMSSTp.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. Rest-mcts*: Llm self-training via process reward guided tree search. *Advances in Neural Information Processing Systems*, 37:64735–64772, 2024.

Jian Zhang, Zhiyuan Wang, Zhangqi Wang, Yu He, Haoran Luo, Li Yuan, Lingling Zhang, Rui Mao, Qika Lin, and Jun Liu. Maxs: Meta-adaptive exploration with llm agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 13713–13731, 2026a.

Xuan Zhang, Ruixiao Li, Zhijian Zhou, Long Li, Yulei Qin, Ke Li, Xing Sun, Xiaoyu Tan, Chao Qu, and Yuan Qi. Count counts: Motivating exploration in LLM reasoning with count-based intrinsic rewards. In *The Fourteenth International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=9xIBbfItGP.

Yifan Zhang, Giridhar Ganapavarapu, Srideepika Jayaraman, Bhavna Agrawal, Dhaval Patel, and Achille Fokoue. Spiral: Symbolic llm planning via grounded and reflective search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 36527–36535, 2026c.

Ziqi Zhao, Zhaochun Ren, Jiahong Zou, Liu Yang, Zhiwei Xu, Xuri Ge, Zhumin Chen, Xinyu Ma, Daiting Shi, Shuaiqiang Wang, et al. Reinforced efficient reasoning via semantically diverse exploration. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 47994–48007, 2026.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*, 2023.

A Addition descriptions for our method

A.1 Algorithmic Description of our method

The complete LPC-MCTS procedure is summarized in Algorithm 1. The learned count enters the search policy, while terminal outcomes provide the learning signal for policy optimization. The CFN head is trained online and retained throughout policy training, sharing the evidence accumulated across prompts.

A.2 Policy Optimization Details

Unless stated otherwise, the following choices are shared by all compared search variants.

Bounded MCTS iterations. Rollout lengths and tool-interaction costs can vary substantially across prompts, so unconstrained tree search may lead to excessively long rollouts and unpredictable training time. To keep the computational cost bounded, we set a maximum number of MCTS iterations, denoted by T, for each prompt. Once this limit is reached, we stop expanding the search tree and proceed to policy optimization using the trajectories collected so far. We set this practical runtime budget to $T = 24$ during training and $T = 128$ during evaluation.

Task- and model-specific advantage adaptation. For some model–task combinations, including Qwen2.5-7B on tool-integrated mathematical reasoning, we observe that RL can weaken tool-use behavior acquired during SFT. We therefore apply a small, bounded format-aware adjustment to the policy advantage to encourage valid tool use, with scale $\delta = 0.1$. This training adaptation is separate from the LPC-MCTS exploration rule.

Algorithm 1 LPC-MCTS training with learned pseudo-counts

Input: Policy π_θ, prompt dataset $\mathcal{D}$, verifier R, training iterations N, candidate limit $K_{\max} \geq B$, selected batch size B.

Initialize CFN head f_ϕ randomly.
for $i = 1$ to N **do**
  Sample a prompt batch $\mathcal{D}_i$ from $\mathcal{D}$.
  Initialize the policy-training set $\mathcal{D}_{\text{tree}} \leftarrow \varnothing$.
  for each prompt $x \in \mathcal{D}_i$ **do**
    Initialize search-tree root v_0 and terminal pool $\mathcal{C}_x \leftarrow \varnothing$.
    while $|\mathcal{C}_x| < K_{\max}$ **do**
      Select a path from v_0 using Eq. 8, and expand its frontier node with $\pi_{\theta_{\text{old}}}$.
      Continue tree expansion until obtaining a terminal trajectory τ.
      During the rollout, construct decision nodes with the entropy rule and compute their pseudo-counts using Eq. 7.
      Evaluate $R(\tau)$, update the tree statistics, and add τ to $\mathcal{C}_x$.
      Store the collected representations and newly sampled Rademacher targets, and periodically update f_ϕ using the newly collected samples.
    end while
    Select the B least-covered trajectories from $\mathcal{C}_x$ using Eq. 11.
    Compute b_x and $A(v, a)$ using Eq. 9, and add the selected action segments to $\mathcal{D}_{\text{tree}}$.
  end for
  Update π_θ with the clipped objective in Eq. 10.
end for

Output: Optimized policy π_θ and online CFN head f_ϕ.

Clipping and loss aggregation. Search trajectories are generated by $\pi_{\theta_{\text{old}}}$ and then reused to update π_θ. As the two policies diverge, large importance ratios can produce unstable gradients, especially for actions with negative advantages. We control these updates with the asymmetric clipping ranges $\epsilon_{\text{low}} = 0.2$ and $\epsilon_{\text{high}} = 0.28$ in Eq. 10, together with a dual-clipping floor $c_{\text{dual}} = 10$ for negative advantages (Ye et al., 2020). Because trajectories have different lengths, we average the surrogate terms over all trainable model-generated tokens in the batch so that longer trajectories do not receive disproportionate weight. Tool-response tokens are environment outputs rather than policy actions and are therefore excluded from optimization. We use neither a KL penalty nor an entropy bonus.

Trajectory candidate selection. LPC-MCTS assigns every completed trajectory in $\mathcal{C}_x$ the mean node pseudo-count in Eq. 11 and ranks the full candidate pool in ascending order. All node scores used by this ranking are recorded under the candidate pool's shared reference state. The first B trajectories form $\mathcal{B}_x$ and provide the trainable samples for policy optimization. For tool-integrated mathematical reasoning, we set $K_{\max} = 12$ and $B = 8$; for search task, we set $K_{\max} = 8$ and $B = 5$.

A.3 Node Construction and Token Aggregation

As introduced in Section 3.2, w denotes the number of newly generated probe tokens used to decide whether to create a node after a tool response. The coefficient $\alpha > 1$ is the multiplicative threshold that compares the probe window's mean token entropy with that of the preceding context: a new node is created when the former exceeds α times the latter. We use $w = 16$ and $\alpha = 2.0$.

Token filtering and aggregation. The CFN encoder receives only the current node segment. Its score mask excludes prompt, padding, and tool-response positions. Among eligible positions within each node, we set $p = 30$ and retain the top $p\%$ by $u_{v,t}$, then partition their indices into maximal contiguous runs. Every run of at least two tokens is kept, and I_v is the union of the tokens in all such runs. If no run reaches the minimum length, the highest-scoring eligible token is used as a singleton fallback. We average the inverse-count scores of all tokens in I_v to obtain $U_\phi(v)$, then apply Eq. 7.

A.4 CFN Training and Shared-Backbone Execution

Representations collected by the search worker are paired with Rademacher targets and processed in vectorized minibatches. Eligibility masks are applied before pairs enter the training set, so environment outputs are not encoded as random-label training examples. The lightweight linear head is optimized with the CFN regression objective and evaluated over token-level representations collected within nodes in vectorized batches.

Each completed simulation contributes the eligible representations along its selected path, paired with independently sampled Rademacher targets. New samples are accumulated between updates and used to update the CFN head every 4 MCTS iterations. The head is initialized once at the beginning of policy training; its parameters persist across prompts and policy-training steps. Its pseudo-counts therefore reflect the policy model's accumulated evidence over the entire training process.

Shared-backbone execution. The CFN and policy share the same LLM backbone. During tree search, SGLang's prefix KV cache is reused to avoid re-encoding cached context. Consequently, CFN scoring adds almost no additional backbone computation, and only the lightweight `hidden_size → coin_flip_dim` head is stored separately.

B Proof of the regret cost of fragmented counts

We prove Theorem 1 in this section. Our finite-time bound adapts the classical UCB analysis (Auer et al., 2002), while our asymptotic argument follows the proof strategy of Fan & Glynn (2022). We extend these arguments to shared exploration counts while retaining separate action-level empirical means. This requires controlling estimation errors using each action's own samples and establishing consistency of the maximum empirical mean within each counting unit.

B.1 Setup and a common form of the two algorithms

We use the action groups defined in Section 4: each group $\mathcal{C}_c$ contains m_c distinct actions with the same reward distribution. Let $M = \sum_{c=1}^{K} m_c$ be the total number of actions. For each $a \in \mathcal{C}_c$, let $(X_{a,n})_{n\geq 1}$ be its potential reward sequence, with $X_{a,n} \sim P_c$, P_c supported on $[0,1]$, and mean μ_c. All these reward variables are mutually independent. The n-th selection of a reveals $X_{a,n}$. Write

$$\overline{X}_{a,n} = \frac{1}{n}\sum_{i=1}^{n} X_{a,i}, \qquad \widehat{\mu}_a(t) = \overline{X}_{a,N_a(t)}.$$

Both algorithms select every action once during the first M rounds and then follow the UCB rules in Section 4. All counts and empirical means refer to the respective algorithm's own history. The shared-count rule is given the action grouping, but neither rule is given the reward means or the identity of the optimal action. Each action's empirical mean uses only its own observed rewards.

To analyze both rules together, call a set of actions that share one exploration counter a *counting unit*. Under independent-count UCB, each action forms its own unit; under shared-count UCB, all actions in $\mathcal{C}_c$ form one unit. Thus, an action group contains m_c counting units under the first rule and one under the second. For a counting unit $g \subseteq \mathcal{C}_c$, define

$$S_g(t) = \sum_{a\in g} N_a(t), \qquad G_g(t) = \max_{a\in g} \widehat{\mu}_a(t). \tag{13}$$

The largest index in g is

$$J_g(t) = G_g(t-1) + \sqrt{\frac{2\log t}{S_g(t-1)}}. \tag{14}$$

Thus, each algorithm selects a unit maximizing $J_g(t)$ and then an action attaining $G_g(t-1)$. This is an exact rewriting of the action-level rule, not a replacement of local values by a pooled estimate. Let

$$\rho_c^{\mathrm{ind}} = m_c, \qquad \rho_c^{\mathrm{share}} = 1$$

denote the number of counting units in action group c. The unique optimal action, denoted $\star$, forms a unit by itself under both rules.

B.2 A FINITE-TIME BOUND

For $B \in \{\mathrm{ind}, \mathrm{share}\}$, every $T \geq M$, every suboptimal action group c, and every $\varepsilon \in (0, \Delta_c)$, we first show that

$$\mathbb{E}_B[N_c(T)] \leq m_c + \rho_c^B \left\lceil \frac{2\log T}{(\Delta_c - \varepsilon)^2} \right\rceil + \frac{m_c}{e^{2\varepsilon^2} - 1} + 2. \tag{15}$$

At round $t > M$, let

$$E_t = \left\{ \widehat{\mu}_\star(t-1) + \sqrt{\frac{2\log t}{N_\star(t-1)}} < \mu_1 \right\}$$

be the event that the optimal index underestimates its mean. By Hoeffding's inequality (Hoeffding, 1963) and a union bound over the possible sample counts,

$$\begin{aligned} \Pr(E_t) &\leq \sum_{n=1}^{t-1} \Pr\left(\overline{X}_{\star,n} < \mu_1 - \sqrt{\frac{2\log t}{n}} \right) \\ &\leq (t-1)t^{-4} \leq t^{-3}. \end{aligned} \tag{16}$$

In particular, $\sum_{t=M+1}^{\infty} \Pr(E_t) < 2$. The union bound handles adaptive sampling without assuming that an empirical mean is independent of its random sample count.

Fix c, T, and ε, and set

$$L_c = \left\lceil \frac{2\log T}{(\Delta_c - \varepsilon)^2} \right\rceil.$$

Suppose action $a \in g \subseteq \mathcal{C}_c$ is selected at $t \leq T$ with $S_g(t-1) \geq L_c$. Its exploration bonus is at most $\Delta_c - \varepsilon$. Outside E_t, its index must be at least μ_1, so

$$\widehat{\mu}_a(t-1) \geq \mu_c + \varepsilon. \tag{17}$$

Every such selection increases the selected action's own count. Therefore, each pair (a, n), where $n = N_a(t-1)$, can account for at most one selection. The expected number of selections satisfying Eq. equation 17 is consequently bounded by

$$\begin{aligned} \sum_{a \in \mathcal{C}_c} \sum_{n=1}^{T-1} \Pr(\overline{X}_{a,n} \geq \mu_c + \varepsilon) &\leq m_c \sum_{n=1}^{\infty} e^{-2n\varepsilon^2} \\ &= \frac{m_c}{e^{2\varepsilon^2} - 1}. \end{aligned} \tag{18}$$

There are m_c initialization selections and at most $\rho_c^B L_c$ subsequent selections with the counting unit's count below L_c. Adding these terms and the expected number of E_t events proves Eq. equation 15. Crucially, every concentration bound uses the selected action's actual local sample size; the shared count is never treated as the sample size of a local empirical mean.

Taking $\varepsilon = \Delta_c/2$, using $e^x - 1 \geq x$ and $\lceil x \rceil \leq x + 1$, and applying the regret decomposition in Section 4 gives

$$R_B(T) \leq \sum_{c=2}^{K} \left[\frac{8\rho_c^B \log T}{\Delta_c} + \frac{2m_c}{\Delta_c} + (m_c + \rho_c^B + 2)\Delta_c \right]. \tag{19}$$

Hence, for each action group, the dependence improves from $m_c \log T$ to $\log T + m_c$, up to gap-dependent factors. The additive dependence on m_c accounts for initialization and errors in the separate empirical values. Comparing these upper bounds alone would not establish an actual regret separation; we prove the matching asymptotic statements next.

B.3 EXACT ASYMPTOTICS

Proof of Theorem 1. **Every counting unit is selected infinitely often.** Suppose some unit has a finite limiting count. Since its count is integer-valued and nondecreasing, it eventually stops being selected. For every unit, $G_g(t) \in [0, 1]$, and

$$\frac{J_g(t)}{\sqrt{2\log t}} = \frac{G_g(t-1)}{\sqrt{2\log t}} + \frac{1}{\sqrt{S_g(t-1)}}.$$

This expression has a positive limit for any unit with finite limiting count, and tends to zero for a unit whose count diverges. There are finitely many units, so all sufficiently late maximizing indices would belong to units that have supposedly stopped being selected, a contradiction. Thus $S_g(t) \to \infty$ for every unit.

The maximum local mean in each counting unit is consistent. Work on the probability-one event on which the strong law holds for every action's potential reward sequence. In a unit $g \subseteq \mathcal{C}_c$, at least one action is selected infinitely often, and the empirical mean of every such action converges to μ_c. Actions selected finitely often eventually have frozen empirical means. No frozen mean can exceed μ_c: otherwise, after all finitely selected actions have received their last samples and all infinitely selected actions have means below the largest frozen mean, the next selection of g would have to select a supposedly frozen action. This contradicts the definition of those actions. Therefore,

$$G_g(t) \longrightarrow \mu_c \qquad \text{almost surely.} \tag{20}$$

This argument permits some actions to receive only finitely many samples; it relies on selecting a maximum-mean action whenever its unit is chosen.

Each suboptimal counting unit incurs one logarithmic sampling cost. Fix a unit $g \subseteq \mathcal{C}_c$, $c \geq 2$, and $\eta \in (0, \Delta_c/2)$. By Eq. equation 20, for all sufficiently large t, $G_g(t-1) \leq \mu_c + \eta$ and $G_\star(t-1) \geq \mu_1 - \eta$. If g is selected at such a round, comparison with the optimal index gives

$$\mu_c + \eta + \sqrt{\frac{2\log t}{S_g(t-1)}} \geq \mu_1 - \eta, \qquad S_g(t-1) \leq \frac{2\log t}{(\Delta_c - 2\eta)^2}.$$

Applying this inequality at the last selection of g up to T yields

$$\limsup_{T\to\infty} \frac{S_g(T)}{\log T} \leq \frac{2}{(\Delta_c - 2\eta)^2} \qquad \text{almost surely.} \tag{21}$$

Since the number of suboptimal counting units is finite, $N_\star(T) = T - O(\log T)$ almost surely.

Let $\tau(T)$ be the last round up to T at which the optimal action is selected. Every subsequent round is suboptimal, so $\tau(T) = T - O(\log T)$ and $\log \tau(T)/\log T \to 1$. At round $\tau = \tau(T)$, the optimal index dominates $J_g(\tau)$, and hence

$$\sqrt{\frac{2\log \tau}{S_g(\tau-1)}} \leq G_\star(\tau-1) - G_g(\tau-1) + \sqrt{\frac{2\log \tau}{N_\star(\tau-1)}} = \Delta_c + o(1). \tag{22}$$

The final bonus tends to zero because the optimal action receives $\tau - O(\log \tau)$ samples. Using $S_g(T) \geq S_g(\tau-1)$ therefore gives

$$\liminf_{T\to\infty} \frac{S_g(T)}{\log T} \geq \frac{2}{\Delta_c^2}.$$

Letting $\eta \downarrow 0$ through a countable sequence in Eq. equation 21 proves

$$\frac{S_g(T)}{\log T} \longrightarrow \frac{2}{\Delta_c^2}, \qquad \frac{N_c^B(T)}{\log T} \longrightarrow \frac{2\rho_c^B}{\Delta_c^2} \qquad \text{almost surely.} \tag{23}$$

Passing to expected regret. For each fixed $\varepsilon \in (0, \Delta_c)$, Eq. equation 15 implies

$$\limsup_{T\to\infty} \frac{\mathbb{E}_B[N_c(T)]}{\log T} \leq \frac{2\rho_c^B}{(\Delta_c - \varepsilon)^2}.$$

Letting $\varepsilon \downarrow 0$ bounds this limsup by $2\rho_c^B/\Delta_c^2$. Conversely, Fatou's lemma and Eq. equation 23 yield

$$\liminf_{T\to\infty} \frac{\mathbb{E}_B[N_c(T)]}{\log T} \geq \mathbb{E}_B\left[\liminf_{T\to\infty} \frac{N_c(T)}{\log T}\right] = \frac{2\rho_c^B}{\Delta_c^2}.$$

Thus the expected counts have the same limiting coefficients. Substitution into the regret decomposition in Section 4, with $\rho_c^{\mathrm{ind}} = m_c$ and $\rho_c^{\mathrm{share}} = 1$, proves the theorem. □

The resulting separation. The theorem compares actual expected regrets, giving

$$R_{\mathrm{ind}}(T) - R_{\mathrm{share}}(T) = 2\log T \sum_{c=2}^{K} \frac{m_c - 1}{\Delta_c} + o(\log T), \tag{24}$$

and

$$\lim_{T\to\infty} \frac{R_{\mathrm{ind}}(T)}{R_{\mathrm{share}}(T)} = \frac{\sum_{c=2}^{K} m_c/\Delta_c}{\sum_{c=2}^{K} 1/\Delta_c}. \tag{25}$$

If every suboptimal action group contains m actions, this ratio is m. Deterministic reward distributions satisfy the assumptions, so the separation can occur even when all local empirical values are exact after initialization.

C Count Fragmentation in Recursive Tree Search

We extend the bandit analysis to recursive UCT with terminal-reward backup. Our proof follows the recursive strategy used in UCT analysis (Kocsis & Szepesvári, 2006) and the subtree-regret decomposition of Coquelin & Munos (2007): we control the error in parent-edge value estimates through the regret accumulated in child subtrees. We adapt this strategy to shared exploration counts while retaining separate empirical action values. In the fixed deterministic tree considered here, this yields matching regret asymptotics that quantify the cost of count fragmentation along the optimal path.

Tree setting. Consider a fixed finite rooted tree $\mathcal{T}$ with root v_0. Each leaf ℓ has a deterministic reward $r_\ell \in [0,1]$, and $V^*(v)$ denotes the largest leaf reward reachable from node v. At each internal node v, divide the actions into groups $\mathcal{C}_{v,c}$, with $m_{v,c} = |\mathcal{C}_{v,c}|$ actions in each group. Actions in the same group lead to separate subtrees with identical structure and matching leaf rewards. Writing v_a for the child reached by action a, their common optimal continuation value is $\mu_{v,c} = V^*(v_a)$ for $a \in \mathcal{C}_{v,c}$.

Assume every internal node has a unique optimal action $a^*(v)$, whose group, indexed by $c^*(v)$, contains only that action. For each suboptimal action group, define $\Delta_{v,c} = V^*(v) - \mu_{v,c} > 0$. Let P^* denote the internal nodes on the unique optimal root-to-leaf path.

Selection and backup. Each simulation recursively selects a root-to-leaf path and backs up the terminal reward along that path. Let $N(v)$ and $N(v,a)$ be the numbers of completed simulations through node v and edge (v,a), respectively, and define $N(v,c) = \sum_{a\in\mathcal{C}_{v,c}} N(v,a)$. The empirical value $\widehat{Q}(v,a)$ averages only the terminal rewards observed through edge (v,a).

At each node, both algorithms first try every action once and then select an action maximizing the corresponding index:

$$\begin{aligned} I_{\mathrm{ind}}(v,a) &= \widehat{Q}(v,a) + \sqrt{\frac{2\log N(v)}{N(v,a)}}, \\ I_{\mathrm{share}}(v,a) &= \widehat{Q}(v,a) + \sqrt{\frac{2\log N(v)}{N(v,c(a))}}. \end{aligned} \tag{26}$$

Thus, both algorithms retain separate empirical action values and use the same terminal-reward backup procedure; only their exploration counts differ.

Following the cumulative-regret formulation for tree search (Coquelin & Munos, 2007), define

$$\mathcal{R}(T) = \sum_{t=1}^{T} \big(V^*(v_0) - r_{\ell_t}\big), \tag{27}$$

where ℓ_t is the leaf reached by simulation t.

Theorem 2 (Regret penalty of count fragmentation in trees)**.** *Under the setting above, for the fixed tree as $T \to \infty$,*

$$\begin{aligned}\mathcal{R}_{\mathrm{ind}}(T) &= 2\log T \sum_{v\in P^*}\sum_{c\neq c^*(v)} \frac{m_{v,c}}{\Delta_{v,c}} + O_{\mathcal{T}}(\log\log T),\\ \mathcal{R}_{\mathrm{share}}(T) &= 2\log T \sum_{v\in P^*}\sum_{c\neq c^*(v)} \frac{1}{\Delta_{v,c}} + O_{\mathcal{T}}(\log\log T).\end{aligned} \tag{28}$$

If there is at least one suboptimal action group on P^* and every such group contains m actions, then $\mathcal{R}_{\mathrm{ind}}(T)/\mathcal{R}_{\mathrm{share}}(T) \to m$. The constants in the remainder may depend on the fixed tree.

Proof. We use local visit counts and proceed by induction on subtree height. Each subtree is analyzed in its own local visit time. Its search process can be continued beyond the visits made by the parent, so the bounds also apply to any finite observed prefix.

Recursive identities. For a node v, let n denote its number of visits and let $N_{v,a}(n)$ count selections of action a during those visits. Define the local subtree regret as

$$\mathcal{R}_v(n) = nV^*(v) - \sum_{i=1}^{n} r_{\ell_i^{(v)}}, \qquad \mathcal{R}_v(0) = 0, \tag{29}$$

where $\ell_i^{(v)}$ is the leaf reached on the ith visit to v. For a leaf, $\mathcal{R}_v(n) = 0$.

Since every visit to child v_a passes through edge (v, a), the empirical edge value satisfies, for $k = N_{v,a}(n) > 0$,

$$\widehat{Q}_v(a; n) = V^*(v_a) - \frac{\mathcal{R}_{v_a}(k)}{k}. \tag{30}$$

Thus, the error in the parent edge value is exactly the average regret accumulated within the child subtree.

Writing $\Delta(v, a) = V^*(v) - V^*(v_a)$, we also have

$$\mathcal{R}_v(n) = \sum_{a\in\mathcal{A}(v)} \Delta(v,a) N_{v,a}(n) + \sum_{a\in\mathcal{A}(v)} \mathcal{R}_{v_a}(N_{v,a}(n)). \tag{31}$$

This follows by decomposing each simulation's regret as $V^*(v) - r_\ell = [V^*(v) - V^*(v_a)] + [V^*(v_a) - r_\ell]$.

Induction hypothesis and counting units. Suppose every child subtree u satisfies

$$\mathcal{R}_u(k) = A_u \log k + O_{\mathcal{T}}(\log\log k) \qquad \text{as } k \to \infty. \tag{32}$$

In particular, for a suitable finite constant C_u,

$$0 \le \mathcal{R}_u(k) \le C_u \log(k + \mathrm{e}) \qquad \text{for all } k \ge 0. \tag{33}$$

These statements hold at leaves with $A_u = C_u = 0$.

At the parent node v, a *counting unit* is a set of actions sharing one exploration counter. Under independent-count UCT, each action forms its own unit; under shared-count UCT, all actions in $\mathcal{C}_{v,c}$ form one unit. For a counting unit g, suppressing v in the notation, define

$$\begin{aligned} S_g(n) &= \sum_{a\in g} N_{v,a}(n),\\ G_g(n) &= \max_{a\in g} \widehat{Q}_v(a; n),\\ J_g(n) &= G_g(n) + \sqrt{\frac{2\log n}{S_g(n)}}. \end{aligned} \tag{34}$$

Maximizing the action indices is equivalent to choosing a unit maximizing $J_g(n)$ and then an action attaining $G_g(n)$. This is only a rewriting of the selection rule: empirical values remain separate.

Let μ_g be the common optimal child value within unit g, and let $\Delta_g = V^*(v) - \mu_g$. The optimal counting unit, denoted by g_*, contains only the optimal action.

Consistency and a coarse count bound. Every counting unit is visited infinitely often. Otherwise, some units would have eventually constant positive counts s_g. For such units, $J_g(n)/\sqrt{2\log n} \to 1/\sqrt{s_g} > 0$, whereas this normalized index tends to zero for units whose counts diverge. Since there are finitely many units, a maximizing unit would eventually have to belong to the set whose counts supposedly stopped increasing, a contradiction.

We next control the value estimate within each counting unit. By Eq. equation 30, $\widehat{Q}_v(a;n) \le \mu_g$ for every $a \in g$. At least one action in g has count $k \ge S_g(n)/|g|$. Applying Eq. equation 33 to that action gives

$$\begin{aligned} 0 \le b_g(n) &:= \mu_g - G_g(n) \\ &\le D_g \frac{\log(S_g(n)+\mathrm{e})}{S_g(n)}, \end{aligned} \tag{35}$$

where $D_g = |g| \max_{a\in g} C_{v_a}$. Hence $G_g(n) \to \mu_g$. This argument does not require every action within a shared counting unit to be visited infinitely often.

If a suboptimal unit g is selected after n completed visits, then

$$\mu_g + \sqrt{\frac{2\log n}{S_g(n)}} \ge J_g(n) \ge J_*(n) \ge G_*(n).$$

For sufficiently large n, consistency gives $G_*(n) \ge V^*(v) - \Delta_g/2$, and therefore $S_g(n) \le 8\log n/\Delta_g^2$ at such selection times. Applying this bound at the last selection of g and allowing for its unit increment yields $S_g(n) = O_{\mathcal{T}}(\log n)$ at every sufficiently large time. Consequently,

$$S_*(n) = n - O_{\mathcal{T}}(\log n), \qquad b_*(n) = O_{\mathcal{T}}\left(\frac{\log n}{n}\right). \tag{36}$$

Matching upper and lower count bounds. We first sharpen the upper bound. Whenever a suboptimal unit g is selected, comparison with the optimal unit gives

$$\sqrt{\frac{2\log n}{S_g(n)}} \ge \Delta_g - b_*(n).$$

Using Eq. equation 36 and the fixed positive gap Δ_g,

$$S_g(n) \le \frac{2\log n}{(\Delta_g - b_*(n))^2} = \frac{2\log n}{\Delta_g^2} + O_{\mathcal{T}}\left(\frac{(\log n)^2}{n}\right).$$

Applying this inequality at the last selection of g, including its unit increment, proves

$$S_g(n) \le \frac{2\log n}{\Delta_g^2} + O_{\mathcal{T}}(1). \tag{37}$$

For the lower bound, let $\tau \le n$ be the last visit on which the optimal action was selected, and put $s = \tau - 1$. All visits after τ are suboptimal, so the coarse count bound implies $s = n - O_{\mathcal{T}}(\log n)$ and $\log s = \log n + o(1)$. At the decision after s completed visits, $J_*(s) \ge J_g(s)$. Since $G_*(s) \le V^*(v)$,

$$\sqrt{\frac{2\log s}{S_g(s)}} \le \Delta_g + b_g(s) + \sqrt{\frac{2\log s}{S_*(s)}}. \tag{38}$$

The last two terms tend to zero, so $S_g(s) \ge c_g \log s$ for some constant $c_g > 0$. Together with Eq. equation 37, this gives $S_g(s) = \Theta_{\mathcal{T}}(\log s)$ at these comparison times. Equation equation 35 then yields

$$b_g(s) = O_{\mathcal{T}}\left(\frac{\log\log s}{\log s}\right), \qquad \sqrt{\frac{2\log s}{S_*(s)}} = O_{\mathcal{T}}\left(\sqrt{\frac{\log s}{s}}\right).$$

Substituting these estimates into Eq. equation 38 and expanding the reciprocal square around the fixed gap Δ_g gives

$$S_g(s) \ge \frac{2\log s}{\Delta_g^2} - O_{\mathcal{T}}(\log\log s).$$

Indeed, the first error contributes $O(\log\log s)$ after multiplication by $\log s$, while the second contributes $O((\log s)^{3/2}/\sqrt{s}) = o(1)$. Using $S_g(n) \geq S_g(s)$ and $\log s = \log n + o(1)$ proves

$$S_g(n) \geq \frac{2\log n}{\Delta_g^2} - O_{\mathcal{T}}(\log\log n). \tag{39}$$

Combining the two bounds,

$$S_g(n) = \frac{2\log n}{\Delta_g^2} + O_{\mathcal{T}}(\log\log n). \tag{40}$$

From local counts to tree regret. For algorithm $B \in \{\mathrm{ind}, \mathrm{share}\}$, define $\rho_{v,c}^{\mathrm{ind}} = m_{v,c}$ and $\rho_{v,c}^{\mathrm{share}} = 1$. Independent counting has one counting unit per action, whereas shared counting has one per action group. Summing Eq. equation 40 within each suboptimal action group gives

$$N_{v,c}^B(n) = \frac{2\rho_{v,c}^B \log n}{\Delta_{v,c}^2} + O_{\mathcal{T}}(\log\log n). \tag{41}$$

The local action-gap contribution in Eq. equation 31 is therefore

$$2\log n \sum_{c \neq c^*(v)} \frac{\rho_{v,c}^B}{\Delta_{v,c}} + O_{\mathcal{T}}(\log\log n).$$

Let $u_* = v_{a^*(v)}$ be the optimal child. It receives $n_* = n - O_{\mathcal{T}}(\log n)$ visits, so the induction hypothesis gives

$$\mathcal{R}_{u_*}^B(n_*) = A_{u_*}^B \log n + O_{\mathcal{T}}(\log\log n).$$

Every suboptimal child receives at most $O_{\mathcal{T}}(\log n)$ visits. Its additional subtree regret is bounded by

$$\mathcal{R}_{v_a}^B\big(N_{v,a}^B(n)\big) \leq C_{v_a} \log\big(N_{v,a}^B(n) + \mathrm{e}\big) = O_{\mathcal{T}}(\log\log n).$$

This bound also covers actions that receive only finitely many visits.

Substitution into Eq. equation 31 proves

$$\mathcal{R}_v^B(n) = A_v^B \log n + O_{\mathcal{T}}(\log\log n),$$

where

$$A_v^B = A_{u_*}^B + 2 \sum_{c \neq c^*(v)} \frac{\rho_{v,c}^B}{\Delta_{v,c}}. \tag{42}$$

This completes the induction.

Finally, unrolling Eq. equation 42 along the unique optimal path yields

$$A_{v_0}^B = 2 \sum_{v \in P^*} \sum_{c \neq c^*(v)} \frac{\rho_{v,c}^B}{\Delta_{v,c}}.$$

Taking $\rho_{v,c}^{\mathrm{ind}} = m_{v,c}$ and $\rho_{v,c}^{\mathrm{share}} = 1$ proves Eq. equation 28. When every suboptimal action group on P^* contains m actions, the independent-count coefficient is m times the shared-count coefficient, giving the stated regret ratio. □

D Prompt Template and Reward Construction

For both settings, the task instance is placed in the user message, while the prompt specifies the available tool and required function-call format. For each backbone, the corresponding prompt is identical for DAPO, standard MCTS, and LPC-MCTS.

Tool-integrated mathematical reasoning. We use the following system prompt with the `code_interpreter` tool:

Mathematical-reasoning system prompt

You are Qwen, created by Alibaba Cloud. You are a helpful assistant.

Tools

You may call one or more functions to assist with the user query.

You are provided with function signatures within <tools></tools> XML tags:
<tools>
{"type": "function",
 "function": {
  "name": "code_interpreter",
  "description": "A tool for executing code.",
  "parameters": {
   "type": "object",
   "properties": {
    "code": {"type": "string", "description": "The code to execute."}
   },
   "required": ["code"]
  }
 }}
</tools>

For each function call, return a JSON object with function name and arguments within <tool_call></tool_call> XML tags:
<tool_call>
{"name": <function-name>, "arguments": <args-json-object>}
</tool_call>

Search. For Qwen2.5-7B-Instruct, we use the following system prompt with the `search` tool:

Qwen search system prompt

You are Qwen, created by Alibaba Cloud. You are a helpful assistant.

Tools

You may call one or more functions to assist with the user query.

You are provided with function signatures within <tools></tools> XML tags:
<tools>
{"type": "function",
 "function": {
  "name": "search",
  "description": "Searches the web for relevant information based on the given query.",
  "parameters": {
   "type": "object",
   "properties": {
    "query_list": {
     "type": "array",
     "items": {"type": "string"},
     "description": "A list of fully-formed semantic queries. The tool will return search results for each query."
    }
   },
   "required": ["query_list"]
  }
 }}
</tools>

For each function call, return a JSON object with function name and arguments within <tool_call></tool_call> XML tags:
<tool_call>
{"name": <function−name>, "arguments": <args−json−object>}
</tool_call>

For Llama-3.2-3B-Instruct, the serving framework automatically adds the system metadata at runtime. We therefore show only the user-message template below:

Llama search user prompt

Given the following functions, please respond with a JSON for a function call with its proper arguments that best answers the given prompt.

Respond in the format {"name": function name, "parameters": dictionary of argument name and its value}. Do not use variables.

```
{
  "type": "function",
  "function": {
    "name": "search",
    "description": "Searches the web for relevant information based on the given query.",
    "parameters": {
      "type": "object",
      "properties": {
        "query_list": {
          "type": "array",
          "description": "A list of fully−formed semantic queries. The tool will return search
    results for each query.",
          "enum": null
        }
      },
      "required": [
        "query_list"
      ]
    },
    "strict": false
  }
}
```

The model can alternate between reasoning and tool calls, using each tool output to continue its response. Because tool outputs are generated by the environment rather than the policy, their tokens are excluded from both the policy and CFN losses.

During training, we extract the final answer from `\boxed{}` for both tasks. For mathematical reasoning, we normalize the extracted expression before comparing it with the reference answer; for search, we use exact match against the reference answer. We define the base task score as

$$S_{\text{base}} = \begin{cases} +1, & \text{if the final answer is correct,} \\ -1, & \text{otherwise.} \end{cases} \tag{43}$$

To address potential tool-call collapse in some RL settings, we add an auxiliary tool-call reward to DAPO training. For incorrect trajectories, the bonus increases by 0.05 for each interaction turn beyond the first two and is capped at 0.4. Correct trajectories receive a unit bonus when they contain more than two turns.

Table 3: Supervised fine-tuning configurations.

Hyperparameter	Value
Optimizer	AdamW
Learning rate	1×10^{-5}
Weight decay	0.01
Learning-rate schedule	Cosine
Warmup ratio	0.1
Global batch size	16
Maximum sequence length	16,384 (math) / 8,192 (search)
Training epochs	2

Table 4: Policy training configuration for tool-integrated mathematical reasoning.

Hyperparameter	Value
General training hyperparameters	
Backbones	Qwen2.5-7B / Qwen3-4B
Optimizer	AdamW
Policy learning rate	1×10^{-6}
Prompts per training batch	128
Samples per prompt	8
Mini-batch size	128
Max prompt length	1,024
Max response length	18,000
Rollout temperature	1.0
MCTS hyperparameters	
Maximum tree depth	6
Maximum tool-interaction turns per node	3
UCT coefficient c_{count}	2.0
Maximum response length per node	3,072

E Training and Inference Configurations

E.1 Training Configurations

All experiments were conducted on 8 NVIDIA H200 GPUs. Tool-integrated mathematical reasoning uses Qwen2.5-7B and Qwen3-4B, while search uses Qwen2.5-7B-Instruct and Llama-3.2-3B-Instruct.

Supervised Fine-Tuning. Before RL training, we perform SFT on each policy backbone. For tool-integrated mathematical reasoning, Qwen2.5-7B and Qwen3-4B are trained on the ReTool SFT corpus (Feng et al., 2026). For search, Qwen2.5-7B-Instruct and Llama-3.2-3B-Instruct are trained on the same 2,000 examples sampled from the Chain-of-Agents SFT corpus (Li et al., 2025a). The SFT configurations are summarized in Table 3.

Policy training. Tables 4 and 5 summarize the policy training settings for the two tasks. We train each experiment for 100 steps.

CFN configuration. Table 6 summarizes the parameters specific to the CFN estimator. The CFN head is randomly initialized and trained online. During search, newly collected samples are accumulated, and the head is updated after every four MCTS iterations. Backbone representations are treated as fixed inputs to the CFN loss, so these updates do not propagate gradients into the policy backbone.

Table 5: Policy training configuration for search.

Hyperparameter	Value
General training hyperparameters	
Backbones	Qwen2.5-7B-Instruct / Llama-3.2-3B-Instruct
Optimizer	AdamW
Policy learning rate	1×10^{-6}
Prompts per training batch	512
Samples per prompt	5
Mini-batch size	256
Max prompt length	512
Max response length	8,192
Rollout temperature	1.0
MCTS hyperparameters	
Maximum tree depth	4
Maximum tool-interaction turns per node	2
UCT coefficient c_{count}	2.0
Maximum response length per node	2048

Table 6: CFN estimator and online-training configurations.

Hyperparameter	Value
CFN output dimension d	20
Optimizer	AdamW
Online learning rate	1×10^{-4}
CFN-head update frequency	Every 4 MCTS iterations
Token filtering	Top 30%

Training cost. On 8 NVIDIA H200 GPUs, a full training run takes approximately 100 hours for tool-integrated mathematical reasoning and 50 hours for search. Across all evaluated tasks and backbones, LPC-MCTS increases training time by no more than 20% relative to standard MCTS, making it a lightweight addition for exploration through learning the pseudo-count.

E.2 Inference Configurations

For sampling-based evaluation of mathematical reasoning, we use a temperature of 0.6 and top-p sampling with $p = 0.95$. For search, we evaluate greedy decoding and tree-based inference with standard MCTS or LPC-MCTS with a temperature of 0.6 and top-p sampling with $p = 0.95$. For MCTS-based evaluation, we run a single search tree for each problem and collect 16 terminal trajectories from that tree. We report mean@16 as the fraction of these trajectories whose final answers are correct, using the same protocol and search budget for standard MCTS and LPC-MCTS.

E.3 Search Retrieval Configuration

All search experiments use the same offline retrieval service. We use `PeterJinGo/wiki-18-corpus`, a 2018 Wikipedia corpus containing 21,015,324 documents. Queries are encoded on CPU with `intfloat/e5-base-v2` into 768-dimensional embeddings.

Retrieval uses the precomputed `PeterJinGo/wiki-18-e5-index-HNSW64` FAISS HNSW64 index, which contains 21,015,324 vectors and uses inner-product similarity. We set `efSearch` to 128 and return the top three documents for each query.

F Additional Experimental Results

F.1 Coin Flipping Network

F.1.1 Token-level Uncertainty Visualization

For each selected response, we align the raw scalar score $u_{v,t}$ with every eligible token position and preserve the original generation order. Prompt, padding, and tool-response positions are excluded using the same mask as in the node-level estimator. A larger $u_{v,t}$ indicates higher CFN uncertainty due to less accumulated representation evidence.



Figure 3: **An example of token-level uncertainty estimated by CFN within a Qwen3-4B modular-arithmetic response.** Each eligible generated token is colored by its raw CFN score $u_{v,t}$. Darker colors indicate higher uncertainty estimate, while lighter colors indicate relatively lower estimates. Tool responses are shown in gray and excluded from CFN scoring. The same applies hereafter.



Figure 4: An example of token-level uncertainty estimated by CFN within a Qwen3-4B modular-arithmetic response.



Figure 5: An example of token-level uncertainty estimated by CFN within a Qwen2.5-7B modular-arithmetic response.



Figure 6: An example of token-level uncertainty estimated by CFN within a Qwen2.5-7B geometric-reasoning response.

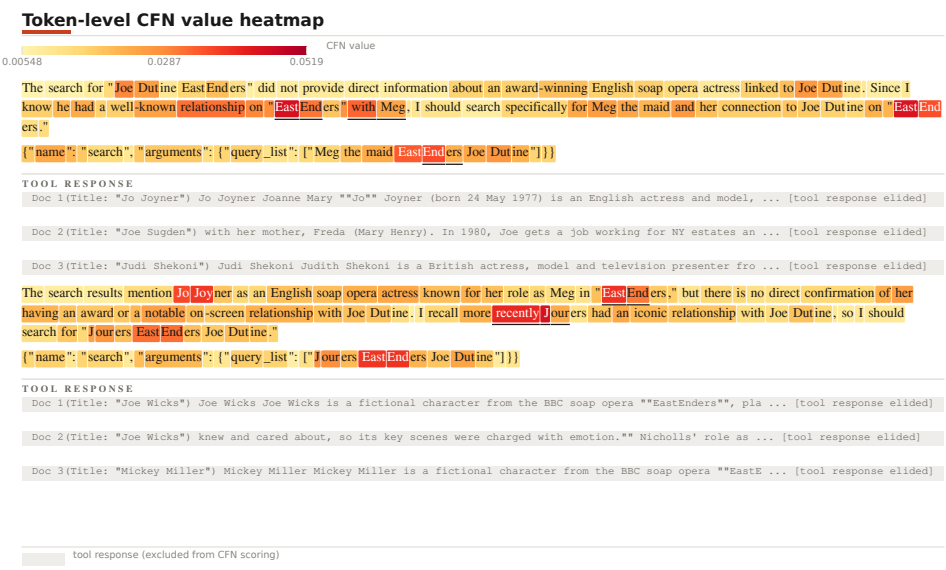


Figure 7: An example of token-level uncertainty estimated by CFN within a Qwen2.5-7B-Instruct search response.



Figure 8: An example of token-level uncertainty estimated by CFN within a Llama-3.2-3B-Instruct search response.

Across the examples, uncertainty changes within each response. In Figure 3, high scores appear when the model changes its plan from direct modulo calculations to a different exponentiation method, and when it builds a new tool call. In Figure 4, high scores appear when the model compares two possible values of p, applies Fermat's little theorem, and later fixes its calculation. In Figure 5, high scores appear when the model works through a sampling process that combines discrete and continuous choices and turns it into simulation code. Figure 6 shows another geometry example: high scores appear when the model notices that its first radius calculation cannot be right, rethinks the geometry, and writes the correct equation using the Pythagorean theorem. In Figure 7, high scores concentrate on named entities and relation-bearing terms as the model reformulates its query after the initial search results do not resolve the target relation. In Figure 8, high scores appear when the model combines retrieved archaeological evidence, reasons over its temporal context, and commits to the final answer despite the absence of a direct match. By contrast, common connecting words, routine calculations, and repeated search scaffolding generally receive lower scores. The high scores usually cover several neighboring tokens rather than isolated tokens. Across both task types, these high-score spans correspond to changes in reasoning strategy, tool-query formulation, evidence synthesis, code, or specialized mathematical steps. This suggests that the learned CFN head captures high-level reasoning patterns in both internal reasoning and tool-augmented information seeking, rather than only token-level differences. Each heatmap has its own score range, so its colors should be read within that example rather than compared directly with the other figures.

F.1.2 Reasoning-Pattern Sensitivity in Rollout Trees

During model rollouts, we record each expanded parent node together with its children and their learned pseudo-counts. We sample some instances from these rollout records for qualitative visualization. Each instance contains two textually different children that follow the same or a closely related reasoning pattern, together with a third child that takes a more distinct approach.

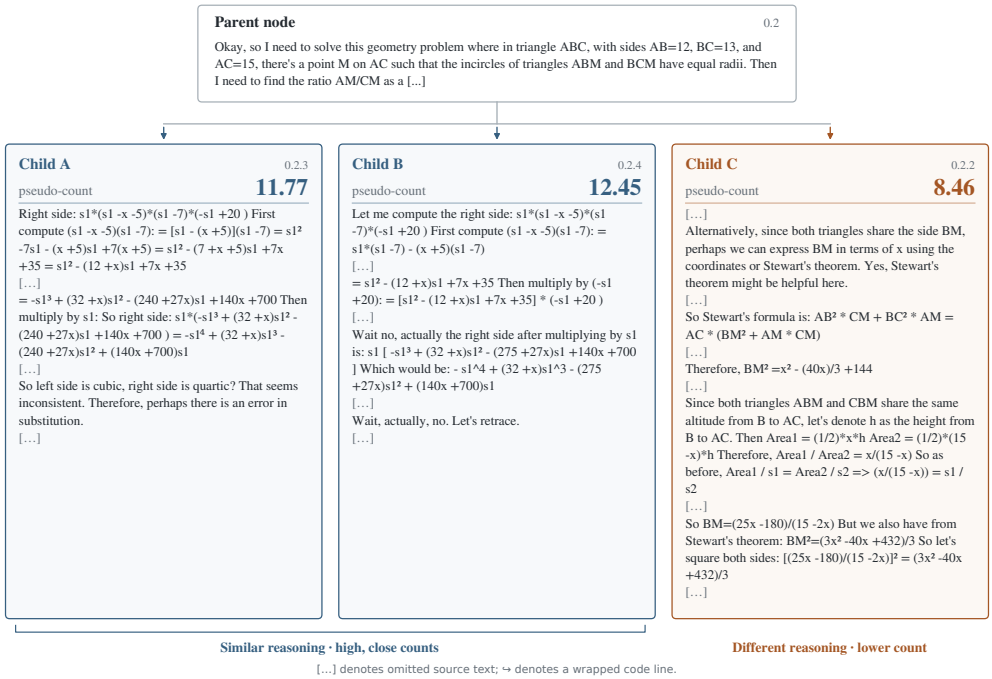


Figure 9: **A geometry case from an actual Qwen3-4B rollout tree.** Children A and B continue the same algebraic expansion and receive close pseudo-counts of 11.77 and 12.45. Child C instead combines Stewart's theorem with an area argument and receives a lower pseudo-count of 8.46.

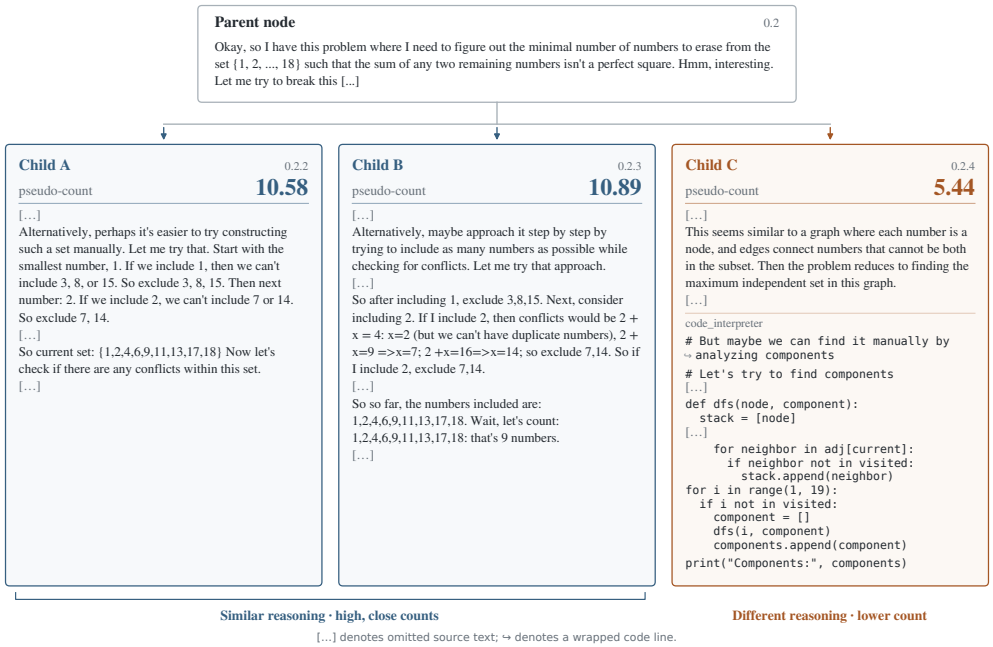


Figure 10: **A combinatorial-reasoning case from an actual Qwen3-4B rollout tree.** Children A and B repeat closely related manual constructions and receive pseudo-counts of 10.58 and 10.89. Child C reformulates the task as a maximum-independent-set problem and receives a lower count of 5.44.

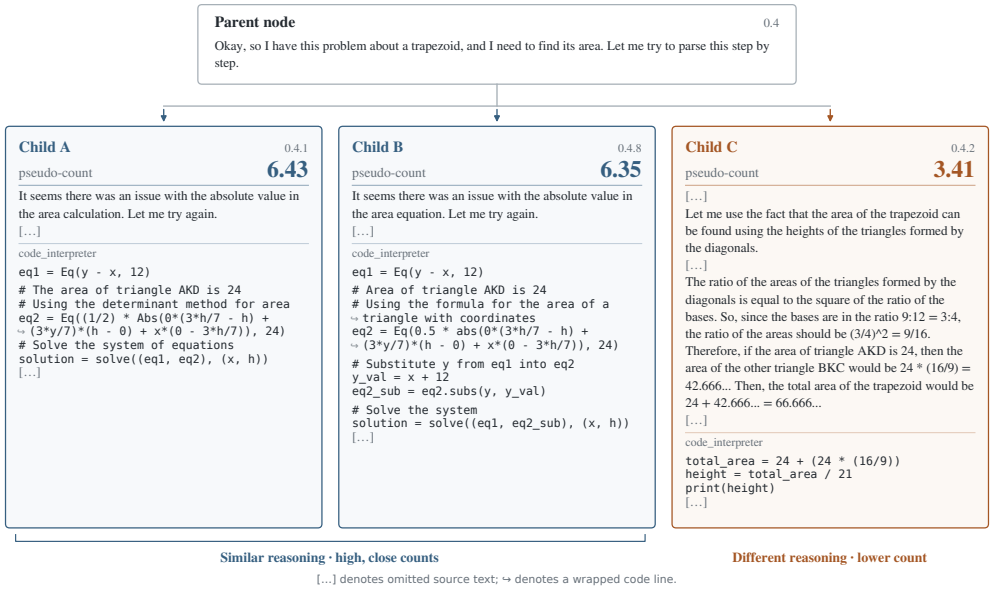


Figure 11: **A geometry case from an actual Qwen2.5-7B rollout tree.** Children A and B retry closely related coordinate-based calculations and receive pseudo-counts of 6.43 and 6.35. Child C replaces this route with an area-ratio argument and receives a lower count of 3.41.

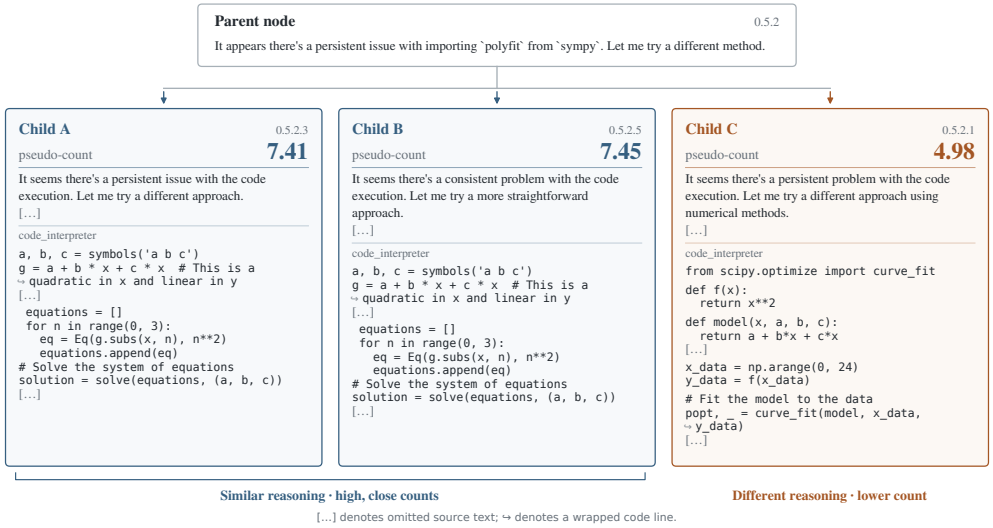


Figure 12: **A symbolic-computation case from an actual Qwen2.5-7B rollout tree.** Children A and B repeat the same symbolic-solving attempt and receive nearly identical pseudo-counts of 7.41 and 7.45. Child C switches to numerical fitting and receives a lower count of 4.98.

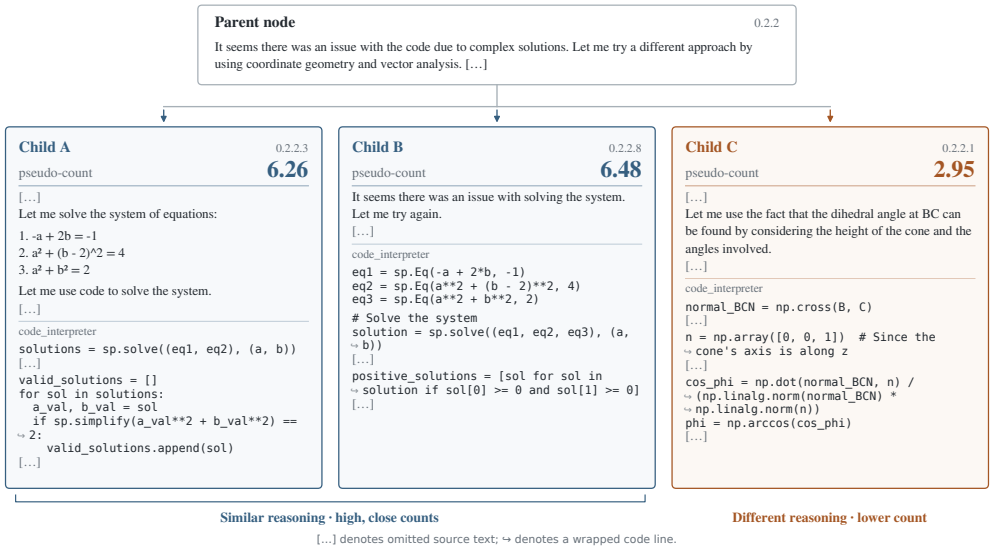


Figure 13: **A solid-geometry case from an actual Qwen2.5-7B rollout tree.** Children A and B retry the same coordinate-system equations and receive pseudo-counts of 6.26 and 6.48. Child C computes the dihedral angle with a vector normal and receives a lower count of 2.95.

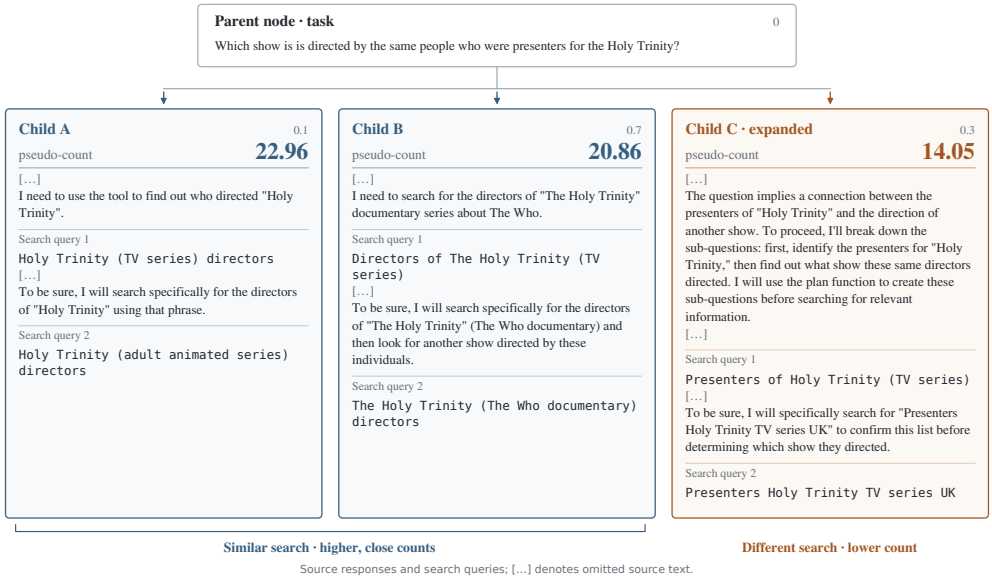


Figure 14: **A query-planning case from an actual Qwen2.5-7B-Instruct search rollout tree.** Children A and B repeatedly search for the program's directors and receive close pseudo-counts of 22.96 and 20.86. Child C instead follows the presenter relation first and receives a lower count of 14.05.

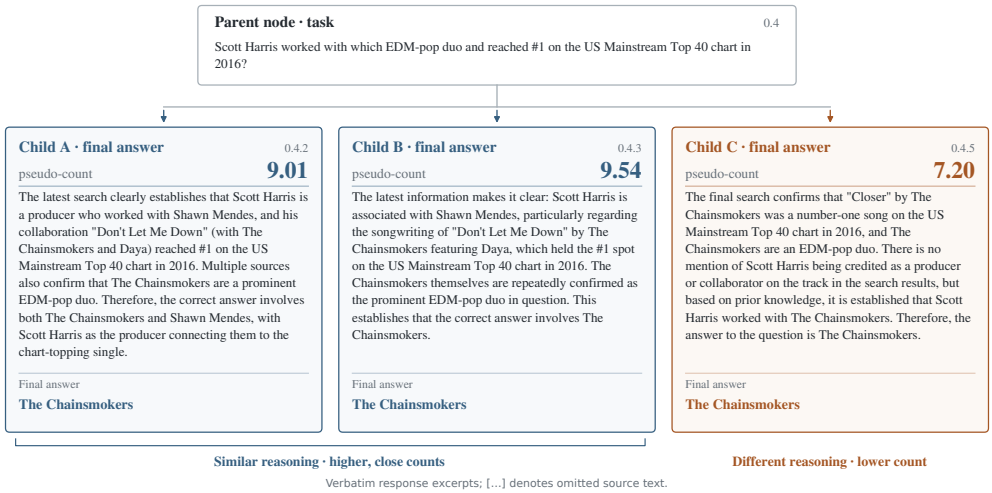


Figure 15: **An evidence-use case from an actual Llama-3.2-3B-Instruct search rollout tree.** Children A and B organize the same final answer around retrieved relationship evidence and receive pseudo-counts of 9.01 and 9.54. Child C relies on prior knowledge after noting an evidence gap and receives a lower count of 7.20.

Together, Figure 1 and Figures 9–15 show this behavior across eight sampled rollout-tree examples. The six mathematical-reasoning cases cover changes in theorem, representation, and computational strategy, while the two search cases cover query planning and evidence use. Across both tasks and model families, textually different children following the same high-level pattern receive close pseudo-counts, whereas a child that changes the reasoning pattern shows a substantially larger pseudo-count gap from the aligned pair. The final search case further shows that this distinction can persist even when all children produce the same answer, indicating that the estimator responds to the reasoning path rather than outcome identity alone.

This behavior addresses a limitation of standard MCTS counts. Standard MCTS maintains an independent tabular count for each child, so sampling child A does not increase the count of a semantically related child B. Consequently, B can still appear unexplored even when its reasoning pattern has already been covered through A. LPC-MCTS instead assigns pseudo-counts from learned hidden representations, allowing such branches to share familiarity. Holding the other UCT terms fixed, the larger counts of A and B reduce redundant exploration of the repeated pattern, while the lower count of C gives the distinct strategy a larger exploration bonus.

F.2 MCTS Training

F.2.1 Training Dynamics

Figure 16 compares the answer scores of standard MCTS and LPC-MCTS across four task–model settings at training time. The curves are smoothed using a 3-step running average.

F.2.2 Ablation Studies

To separate the effects of the training and inference configurations, Table 7 specifies the method used at each stage. Pseudo-count-guided trajectory selection is used only to construct the policy-training batch, so we ablate it only during training by removing the ranking and selection rule in Eq. 11. Token-to-node aggregation is used to compute node pseudo-counts in both stages and is therefore ablated separately during training and inference. When aggregation is removed at a given stage, let t_{last} be the last eligible model-generated token in node v. We directly use its CFN score to compute the node-level pseudo-count:

$$U_\phi(v) = u_{v,t_{\mathrm{last}}}, \qquad C_\phi(v) = \frac{1}{U_\phi(v) + \epsilon}. \tag{44}$$

The combined training variant removes both trajectory selection and node aggregation. We also remove node aggregation at both stages to assess its end-to-end effect. All other settings are kept identical to full LPC-MCTS.

Table 7 shows that both components contribute across backbones. Removing pseudo-count-guided trajectory selection lowers the average by 2.71 points on Qwen2.5-7B and 1.42 points on Qwen3-4B. Removing token-to-node aggregation during training lowers the average by 0.83 and 2.67 points, while removing it only at inference lowers it by 1.37 and 2.00 points. Removing aggregation at both stages produces larger drops of 2.33 and 4.29 points, respectively. Removing both training components also reduces the average by 1.50 and 5.12 points. These consistent average degradations support the roles of trajectory selection in policy training and node aggregation in both training and inference.

F.2.3 Hyperparameters

The default token-filtering ratio in LPC-MCTS is $p = 30$, as described in Appendix A.3. To examine sensitivity to this choice, we additionally evaluate $p = 50$, $p = 70$, and $p = 100$. A larger p retains a broader set of token-level CFN scores, while $p = 100$ retains all eligible tokens before contiguous-run filtering. The same value of p is used during training and inference, with all other configurations fixed.

Table 8 shows that the default $p = 30$ achieves the highest average performance on both backbones. Qwen3-4B drops by 1.96–2.25 points with larger filtering ratios, whereas Qwen2.5-7B varies by less than one point across all settings. These results support $p = 30$ as a strong default while indicating that LPC-MCTS is not highly sensitive to this choice, especially on Qwen2.5-7B. Moreover,

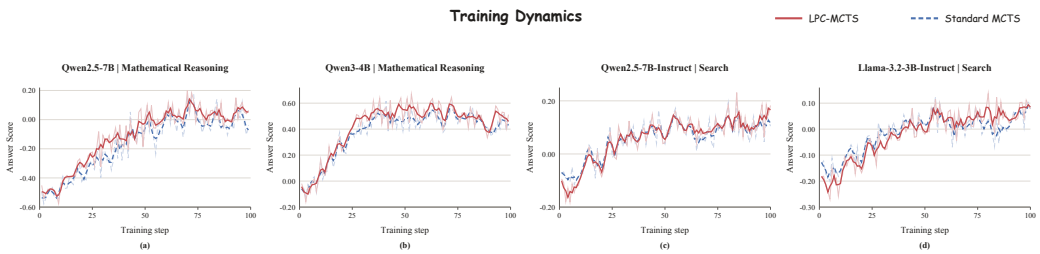


Figure 16: Training reward dynamics of standard MCTS and LPC-MCTS over 100 policy-training steps across two mathematical-reasoning and two search model settings.

Table 7: Ablation results under separate training and inference settings on tool-integrated mathematical reasoning with Qwen2.5-7B and Qwen3-4B. Each value is the mean accuracy over 16 responses per problem (mean@16).

Training	Inference	AIME 24	AIME 25	AIME 26	HMMT25	BRUMO25	Avg.
Qwen2.5-7B							
LPC-MCTS (w/o trajectory selection)	LPC-MCTS	34.79	29.58	27.71	15.83	42.50	30.08
LPC-MCTS (w/o node aggregation)	LPC-MCTS	37.71	28.75	25.63	19.17	48.54	31.96
LPC-MCTS (w/o both)	LPC-MCTS	37.50	26.04	28.96	18.54	45.42	31.29
LPC-MCTS	LPC-MCTS (w/o node aggregation)	35.83	28.33	27.08	19.17	46.67	31.42
LPC-MCTS (w/o node aggregation)	LPC-MCTS (w/o node aggregation)	38.13	27.08	27.08	18.13	41.88	30.46
LPC-MCTS	LPC-MCTS	37.08	30.63	27.92	23.33	45.00	32.79
Qwen3-4B							
LPC-MCTS (w/o trajectory selection)	LPC-MCTS	74.38	63.13	72.08	50.21	65.60	65.08
LPC-MCTS (w/o node aggregation)	LPC-MCTS	75.21	62.71	70.83	48.33	62.08	63.83
LPC-MCTS (w/o both)	LPC-MCTS	70.63	58.13	66.04	47.29	64.79	61.38
LPC-MCTS	LPC-MCTS (w/o node aggregation)	74.79	62.50	70.42	48.96	65.83	64.50
LPC-MCTS (w/o node aggregation)	LPC-MCTS (w/o node aggregation)	73.13	58.75	67.29	49.38	62.50	62.21
LPC-MCTS	LPC-MCTS	78.13	63.54	72.92	51.88	66.04	66.50

Table 8: Sensitivity to the top-$p\%$ token-filtering ratio on tool-integrated mathematical reasoning with Qwen2.5-7B and Qwen3-4B. Each value is the mean accuracy over 16 responses per problem (mean@16).

Top $p\%$	AIME 24	AIME 25	AIME 26	HMMT25	BRUMO25	Avg.
Qwen2.5-7B						
30%	37.08	30.63	27.92	23.33	45.00	32.79
50%	36.25	26.67	28.75	22.71	45.00	31.88
70%	34.38	31.67	27.08	24.79	42.71	32.13
100%	38.96	29.17	29.79	22.08	43.33	32.67
Qwen3-4B						
30%	78.13	63.54	72.92	51.88	66.04	66.50
50%	74.17	61.46	69.58	54.38	63.13	64.54
70%	76.88	59.58	70.00	48.96	66.25	64.33
100%	78.96	60.21	71.25	48.54	62.29	64.25

retaining all eligible tokens ($p = 100$) does not improve the average, suggesting that filtering helps exclude less informative token-level estimates from node aggregation.