

Discover, Then Diversify: Phase-Aware Control of Tree-of-Thought Search

Atkia Mahila¹ Avinash Maurya² M. Mustafa Rafique¹ Bogdan Nicolae²

¹Rochester Institute of Technology, Rochester, NY, USA

²Argonne National Laboratory, Lemont, IL, USA

am9183@rit.edu amaurya@anl.gov mrafique@cs.rit.edu bnicolae@anl.gov

Abstract

Tree-of-Thought (ToT) reasoning has significantly enhanced the problem-solving capabilities of Language Models by enabling the structured exploration of reasoning trajectories. However, the efficiency of these frameworks is often bottlenecked by search algorithms that struggle to navigate vast reasoning spaces at a reasonable inference cost. In this work, we address the challenge of optimizing LLM performance on mathematical reasoning tasks under limited token budgets. A key limitation of existing search strategies is that they apply the same exploration and selection rule throughout the entire search, which under a finite budget, often leads to token exhaustion with suboptimal or no candidate answers. Effective search instead requires different behavior at different stages. We introduce PACE (Phase-Aware Controller for Exploration), a strategy that adjusts its behavior based on the current state of the search: it prioritizes progress toward a first complete answer when none exists, and broadens exploration toward structurally distinct regions later on. Across GSM8K and Math500, two model scales, and token budgets from 3k to 10k, PACE consistently matches or improves over recent state-of-the-art baselines. On GSM8K with Llama-3.2-3B at a 3k budget, PACE delivers roughly a 56% relative accuracy improvement over the strongest tree-search baseline, and an early-stopping variant achieves comparable accuracy using about two-thirds of the tokens.

1 Introduction

Tree-of-Thought (ToT) reasoning has become a common abstraction for inference-time scaling in large language models (Yao et al., 2023). It extends chain-of-thought (Wei et al., 2022) prompting by allowing a model to maintain and explore multiple partial reasoning paths, each at arbitrary depths, rather than committing to a single linear trajectory. In this view, each partial solution corresponds to a node in the search tree, and the set

of expandable nodes at any point forms the search *frontier*. Inference-time reasoning then reduces to an exploration strategy over this frontier. At each step, one frontier node is chosen for expansion, and the choice must be informed, typically by a process reward model (PRM) score (Lightman et al., 2024). Therefore, despite high quality responses from the underlying model, the final outcome depends on *which nodes the exploration strategy expands*, across competing branches.

Within the tree, each branch is composed of multiple nodes. During a reasoning step, the *strategy* determines the candidate branch that should be expanded next (detailed in next paragraph). All the predecessor nodes of a branch are fed as input to a model (typically an LLM), which auto-regressively generates an output sequence, which then acts as the successive node that gets appended to the corresponding reasoning branch. Each token produced during the auto-regressive decode phase accumulates towards the *token budget*, i.e., the number of tokens consumed to solve the target problem.

The *strategy* determines which of the branches should be expanded in the subsequent reasoning step using the following two components. First, it scores all branches using strategies such as PRMs (Lightman et al., 2024), learned value estimates (Xie et al., 2024; Ding et al., 2025), or semantic similarity to existing states (Wang et al., 2025b; Kim et al., 2025), which help it rank, score, or filter partial reasoning states. Second, it uses a search strategy that acts upon the scores to determine the next subset of promising trajectories. While the efficiency of reasoning depends on both scoring heuristics, which can be noisy (Cinquin et al., 2025; Sun et al., 2025; Wang et al., 2025b), and the search strategy, i.e., how well different trajectories are explored, suboptimality in either component may lead to incorrect results or, under limited inference-time token budgets (Snell et al., 2025; Wang et al., 2024a), exhaustion of the budget

before any complete answer is found. For the scope of this work, we focus on studying and improving the latter—when should the search recursively pursue a subset of promising trajectories vs. when should it broaden its exploration across different branches. Concretely, given an inference-time budget, the search problem is to allocate expansions so that the budget produces a useful set of complete candidate answers before exhaustion. The final prediction is then selected from these candidates through voting. Therefore, the search must use its budget both to reach viable candidates and to produce useful alternatives among them.

Limitations of State-of-the-Art. While recent efforts such as DPTS (Ding et al., 2025), SSDP (Kim et al., 2025), and FETCH (Wang et al., 2025b) target a similar scenario as ours, they either focus on semantic mechanisms, such as merging or pruning selected branches, and/or retain a score-guided search objective throughout reasoning irrespective of the reasoning progress and remaining token budget. Although more effective than classical BFS/DFS adopted by earlier ToT approaches (Yao et al., 2023), a score-guided best-first search with a fixed selection rule remains prone to two limitations that we observe empirically: (a) premature over-expansion of shallow nodes is prioritized over finding one potential complete candidate solution, oblivious to remaining token budget; and (b) localization of exploration within a few dominant subtrees, limiting diversity. Section 3 quantifies each of these limitations and highlights that the baseline approach consumes an order of magnitude more tokens than needed by the winning path and concentrates exploration coverage in $\approx 64\%$ of the problems.

Contributions. To tackle the above challenges of fixed-policy search exploration, we propose an adaptive search strategy, PACE (Phase-Aware Controller for Exploration), that balances the exploitation and diversification of ToT. As shown in Figure 1, our search strategy focuses on reaching a (potentially suboptimal but) complete answer first (which results in at least one complete answer under token exhaustion) as shown in the left subfigure labeled “Candidate-seeking” discovery regime, and then dynamically shifts the search objective towards diversifying across candidates and improving the final answer, as shown in the right subfigure of “Candidate-refining” diversification regime. We make the following contributions in this paper:

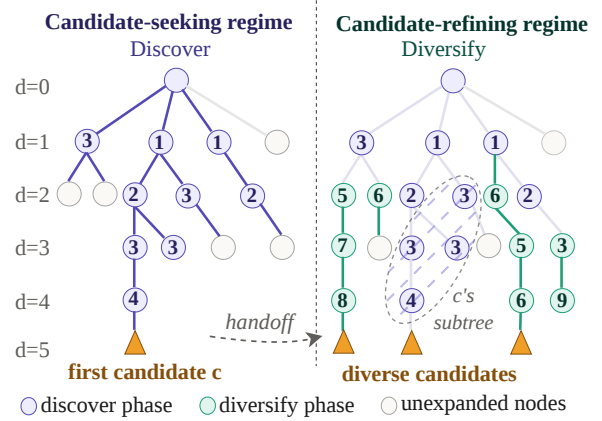


Figure 1: Overview of PACE. Left: in the candidate-seeking regime, selection is biased toward depths where complete trajectories appear. Right: once a first candidate c is found, the strategy pushes selection away from c ’s subtree (hatched) toward structurally distinct regions. Numbered nodes mark expansion steps; nodes sharing a number are expanded in the same round.

1. **Empirical characterization of fixed-policy search strategies.** We identify and quantify two systematic behaviors of PRM-guided best-first search: shallow over-expansion before any candidate exists, and subtree collapse once one does, and use these observations to motivate a regime-conditioned strategy (§3).
2. **A regime-conditioned search policy, PACE, for adaptive inference-time control.** Rather than applying a fixed branch selection rule throughout search, we propose PACE, a Phase-Aware Controller for Exploration (Figure 1), that adjusts the balance between exploration and exploitation according to the search regime (§4).
3. **A principled connection between branching decisions and semantic diversity of continuations.** We frame the branching factor of a node as a function of how semantically varied a small batch of its children is and allocate the token budget accordingly: nodes whose continuations explore distinct reasoning directions are expanded further, while nodes whose continuations repeat are not (§4.3).
4. **Evaluation under fixed token budgets.** We evaluate PACE against four recent inference-time reasoning methods on two mathematical reasoning benchmarks, two model scales, and multiple token budgets. We report how different controllers behave when search is stopped at different resource limits (§6).

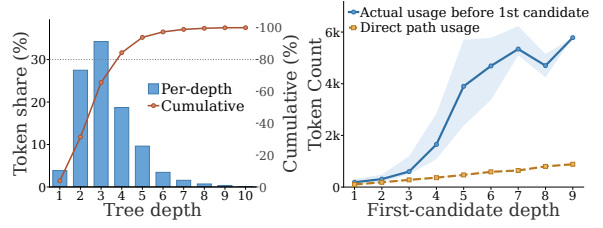
2 Related Work

Structured Inference-Time Reasoning. Chain-of-Thought (CoT) prompting (Wei et al., 2022) and self-consistency (Wang et al., 2023) established that intermediate steps and trajectory aggregation improve reasoning over greedy decoding. Tree of Thoughts (Yao et al., 2023) made coordination across trajectories explicit by recasting inference as a search over partial reasoning states. Extensions of this formulation vary the data structure or control: Graph of Thoughts (Besta et al., 2024) generalizes to arbitrary dependency graphs, and SWAP (Xiong et al., 2025) replaces tree search with planning over an entailment graph.

Search Algorithms and Their Scoring Signals.

Recasting inference as search makes the central question how a finite budget is distributed across competing branches. Controllers have moved from classical BFS/DFS (Yao et al., 2023) to value-function guidance (Wan et al., 2024) and MCTS-style exploration-exploitation rules (Xie et al., 2024; Ding et al., 2025). The common component across these methods is a learned score, typically a process reward model (PRM) (Lightman et al., 2024) that is used to rank partial states at every iteration. However, PRMs often fail to approximate the long-horizon value of a state, with reliability degrading on shallow partial states (Cinquin et al., 2025). Applied uniformly across the search, this scoring rule produces over-exploration of redundant states (Wang et al., 2025b) and trajectory switching between unrelated branches (Ding et al., 2025). Efforts to improve PRMs themselves include ensembling and variance-reduced training (Wang et al., 2025b), and temporal-difference look-ahead for early-step bias (Sun et al., 2025). These target the score itself; our work instead adapts how the controller uses a score whose reliability varies across the search.

Budget-Aware and Adaptive Reasoning. Recent budget-controlled studies show that gains from complex search often shrink when simple baselines are given matched token budgets (Snell et al., 2025; Wang et al., 2024a), motivating evaluation under explicit resource constraints. Some methods prune semantic redundancy online (Kim et al., 2025; Wang et al., 2025b); others reframe decoding as budgeted A* (Chatziveroglou, 2025) or condition behavior on remaining budget (Miyamoto et al., 2026). These regulate the *quantity* of com-



(a) Token share by depth. (b) Pre-candidate cost vs. direct-path cost.

Figure 2: Budget allocation of the PRM-guided best-first search baseline. Left: blue bars show the fraction of generated tokens spent at each tree depth, and the line shows the cumulative fraction. Right: the solid line shows tokens consumed before the first candidate is found, while the dashed line shows the direct root-to-candidate path cost; shaded regions denote the interquartile range.

putation; we instead ask what *kind* of search to perform at the current stage, conditioning selection on the state of the search rather than the budget.

3 Characterizing Budget Expenditure and Candidate Concentration

We analyze a PRM-guided best-first search baseline (Appendix A) on Math500 with Llama-3.2 3B under an 8k token budget, using Math-Shepherd-PRM-7B as the step-level scorer. The search greedily follows PRM scores: at each step, it selects the highest-scoring partial solutions from the current frontier for expansion, scores the resulting continuations, and returns them to the frontier. We track depth, token expenditure, and candidate discovery across expansions to characterize how this fixed search policy allocates its budget.

3.1 Observation 1: Budget Concentrates at Shallow Depths

Figure 2a reports the share of tokens spent at each depth across the problem set. The cumulative share of total token expenditure crosses 80% by depth four. This indicates that the controller does not steadily move deeper as search progresses; instead, it saturates a shallow region of the tree and continues expanding there until the budget is exhausted. Appendix B reports complementary expansion-step statistics, including the mean expanded depth and the 95th-percentile reached depth as a function of consumed budget, which make this saturation behavior explicit.

Figure 2b compares the two quantities as a function of first-candidate depth: the tokens consumed

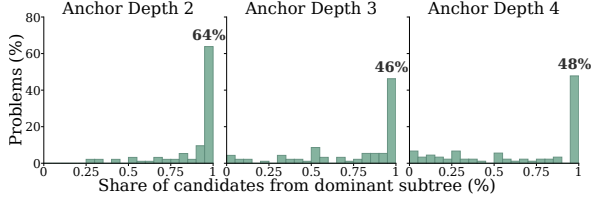


Figure 3: Full of dominant-subtree share $\rho(d^*)$ for anchor depths $d^* \in \{2, 3, 4\}$. Higher values indicate that post-handoff candidates are concentrated within a single early subtree.

before discovering the first candidate, and the cost of the winning path itself (root-to-candidate). Winning-path cost grows roughly linearly with depth; realized pre-candidate cost grows much faster. For example, when the first candidate appears at depth five, the controller consumes roughly 3.9k tokens before discovering it, while the winning path itself costs only about 500 tokens. Under tighter budgets (e.g., 3k tokens), this misallocation can exhaust the budget before any complete candidate appears on deeper problems. Together, these views show that the controller spends most of its budget where productive output does not occur, and the gap widens with problem difficulty, motivating a selection rule that biases toward chain completion before any candidate exists (§4.1).

3.2 Observation 2: Candidate Sets Concentrate in Few Subtrees

Once a candidate answer exists, additional candidates are only useful to the extent that they offer structurally distinct alternatives, since majority voting and related aggregation methods rely on candidates arising from different reasoning paths (Wang et al., 2023). We examine whether the baseline produces such diversity.

To isolate post-candidate behavior, we analyze the baseline after a candidate has already been discovered. For each problem, we cut the search tree at an anchor depth d^* and group all later candidates by the subtree from which they descend, and compute the *dominant-subtree share* $\rho(d^*)$: the fraction of candidates originating from the most frequent subtree. Figure 3 shows strong concentration in a small number of early subtrees. Concretely, at anchor depth $d^* = 2$, 64% of problems have a dominant-subtree share of 100%, meaning that every candidate produced after handoff descends from the same depth-two subtree, and 83% of problems have a dominant-subtree share $\rho(d^*)$ of at least 75%. This concentration persists at deeper

anchors: 65% of problems at $d^* = 3$ and 56% at $d^* = 4$ exceed the 75% threshold. Similar to the shallow case, the fixed selection rule does not account for structural overlap with existing candidates, so the controller might repeatedly sample from the same candidate-producing subtree. This motivates a post-candidate rule that balances exploitation with exploration (§4.2).

3.3 Generality Across Process Reward Models

To examine whether these behaviors depend on the scorer used to guide the search, Appendix D repeats both analyses with Qwen-PRM-7B and observes the same qualitative patterns: budget remains concentrated at shallow depths, and post-candidate solutions remain concentrated in early subtrees. This suggests that the observed behaviors arise from the fixed score-guided search policy rather than from a particular PRM.

Together, these observations identify two phases of search where fixed-policy controllers behave suboptimally: before any candidate exists, by failing to drive expansion toward completion; and after one exists, by failing to diversify away from the candidate-producing region. The next section develops a strategy where the selection rule is conditioned on which phase the search occupies.

4 PACE: A Regime-Conditioned Search Strategy

We propose PACE (Phase-Aware Controller for Exploration) that adapts the search objective as reasoning progresses. Methods that target search redundancy typically do so within a fixed search objective, e.g., by merging or pruning similar semantic states or by stopping early, rather than changing the allocation that causes it. PACE uses two phase-specific objectives: before any candidate is found, it biases selection toward deeper states to secure a complete answer before the budget is exhausted; once a candidate exists, it redirects the remaining budget toward structurally distinct subtrees.

PACE operates over a frontier $\mathcal{F}_t$ of partial reasoning states, a candidate set $\mathcal{C}_t$, and a PRM score $\text{PRM}(v)$ for each frontier node. Section 4.1 and Section 4.2 define the selection rule in the two phases, while Section 4.3 describes the branching rule used during expansion.

4.1 Candidate-Seeking Regime

Before any candidate has been found, the controller has no completed trajectory to reference and must

rely on the PRM score, whose reliability degrades on shallow partial states (Cinquin et al., 2025; Sun et al., 2025). The objective in this regime is to push the search toward depths at which the PRM signal becomes informative and complete trajectories begin to appear. A depth cutoff would enforce this directly, but the appropriate cutoff varies across problems and is unknown in advance; switching to depth-first selection would discard the PRM signal even when it does discriminate. We instead retain the PRM as the primary ranking criterion and add a depth-aware term:

$$S(v) = \text{PRM}(v) + \alpha_t \cdot \text{depth}(v) \quad (1)$$

The coefficient α_t controls how aggressively the rule prefers depth over PRM ranking. We make it responsive to how informative the current frontier scores are: when frontier scores spread, differences between nodes are large, and α_t rises so that the depth term remains large enough to affect the ranking. When frontier scores cluster tightly, the PRM cannot meaningfully separate nodes, and small depth differences are already enough to decide the ranking, so α_t falls. In both cases the depth term retains a comparable influence relative to the range of PRM scores on the frontier. The specific form of α_t as a function of frontier score spread is given in Appendix E.

4.2 Candidate-Refining Regime

Once a candidate c exists, the controller has two new reference points: a reference depth $\text{depth}(c)$ at which complete trajectories appear, and a reference subtree it has already explored. The objective in this regime is to use both to balance exploitation and exploration: remaining near depths where complete trajectories are likely to appear, while pushing search away from an already exploited candidate-producing subtree.

Locality. The first pressure pushes selection away from the subtree of the most recent candidate. A controller that re-selects nodes in c ’s subtree wastes budget on minor variations of an already-completed trajectory. The natural way to recognize “close to c ” would be to compare the semantic content of the reasoning prefixes, e.g., by embedding partial trajectories and computing cosine similarity. This is principled but expensive as every frontier scoring step would require an embedding pass.

We use the tree structure itself as a low-cost proxy. Two nodes that share a long prefix in the

reasoning tree were generated under the same earlier decisions and are likely to remain close in their subsequent reasoning. Two nodes whose lowest common ancestor (LCA) lies near the root diverged early and represent structurally distinct alternatives. We define the *structural overlap* of a frontier node v with the most recent candidate c as:

$$L(v, c) = \frac{\text{depth}(\text{LCA}(v, c))}{\min(\text{depth}(v), \text{depth}(c))} \quad (2)$$

Depth Alignment. Pushed only by the locality term alone, the strategy can drift toward frontier nodes that are structurally distant from c but too shallow to complete a trajectory within the remaining budget. We therefore add a depth-alignment term $D(v)$ that softly favors nodes near $\text{depth}(c)$ and disfavors nodes far from it. The specific functional form is given in Appendix E.

The combined selection rule for the candidate-refining regime is:

$$S(v) = \text{PRM}(v) - \lambda_t L(v, c) + \mu_t D(v) \quad (3)$$

Both λ_t and μ_t respond to the recent behavior of the controller: λ_t rises when recent selections continue to come from inside c ’s subtree and relaxes when they do not; μ_t rises when recent selections drift far from $\text{depth}(c)$ and relaxes when they remain near it. The intent is for the controller to push harder on whichever pressure the current trajectory of the search is failing to respect, rather than applying a fixed weight to either. The exact update rules are given in Appendix E.

4.3 Adaptive Branching

The selection rule decides which parents to expand; branching decides how many children each parent produces. Each child is a continuation of the parent’s partial reasoning, which then roots the subtree of everything that can follow it. Generating more children of a parent therefore commits budget to exploring more of its subtrees, and a fixed branching factor commits the same budget to every parent regardless of what those subtrees look like.

As illustrated in Figure 4, we instead give a parent more branching budget when its children are distinct, and less when they are similar. The underlying assumption is that similar children are likely to lead to similar subtrees, so generating more of them spends budget on largely redundant reasoning. In contrast, diverse children suggest that the parent can lead to different reasoning routes, where

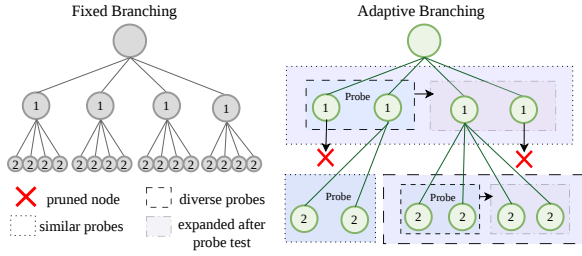


Figure 4: Fixed branching (left) allocates uniform expansion budget. Adaptive branching (right) generates a probe set per parent and uses its spectral diversity to decide whether to expand further (diverse probes) or stop (similar probes).

additional expansion is more useful. Prior work uses semantic similarity to merge or discard redundant continuations (Kim et al., 2025; Wang et al., 2025b); we use the same signal to decide how much branching budget a parent should receive.

The controller first generates a small number of children for each selected parent and measures their similarities. If they are mostly similar, expansion stops at this smaller branching factor. If they are sufficiently distinct, the parent receives additional children up to the maximum branching factor. Thus, more of the expansion budget is assigned to parents whose children indicate multiple reasoning directions. We measure this diversity from the spectral structure of the children’s embedding-similarity matrix; Appendix F gives the formal definition and the full algorithm. After expansion, the same similarity signal is also used to remove redundant siblings before they are inserted into the frontier.

4.4 PACE with Early Stopping

We consider PACE-ES, a variant that combines the same phase-aware selection and adaptive branching rules with an early-stopping criterion. We adapt the early-stopping rule used by DPTS (Ding et al., 2025), described in Appendix G. Once the stopping condition is satisfied, PACE-ES terminates the search instead of consuming the remainder of the nominal token budget. We use this variant to examine the accuracy vs. token consumption tradeoff when consuming the full budget is not required.

5 Experimental Setup

We evaluate PACE against four representative tree-search and structured-reasoning methods on two mathematical reasoning benchmarks, at two model scales, under four fixed token budgets.

Evaluation Datasets. We use two benchmarks: *GSM8K* (Cobbe et al., 2021), a set of 1,319 problems requiring short multi-step arithmetic (typically 2–8 reasoning steps), and *Math500* (Hendrycks et al., 2021), a benchmark spanning diverse difficulty levels and mathematical domains, where higher-difficulty instances require markedly deeper trajectories.

Model Configurations. We use *Llama-3.2-3B-Instruct* and *Llama-3.1-8B-Instruct* as generators (Grattafiori et al., 2024), and *Math-Shepherd-PRM-7B* (Wang et al., 2024b) as the process reward model for the main results, following the same configuration as recent work on PRM-guided tree search (Ding et al., 2025; Kim et al., 2025).

Token Budget. We impose per-problem generation budgets of 3k, 5k, 8k, and 10k output tokens. Budgets are enforced after each batched expansion step. Because expansion may issue multiple generation calls in a single round, the total generated-token count can slightly exceed the nominal budget before the stopping condition is checked.

Evaluation Testbed. We used the ALCF Polaris supercomputer (Argonne Leadership Computing Facility, 2026) to evaluate PACE. Each server has a 32-core 2.8 GHz AMD EPYC Milan CPU, 512 GB of DDR4 memory, and 4× NVIDIA A100 GPUs with 40 GB HBM2. We use up to 64 servers with shared access to a Lustre parallel file system.

5.1 Compared Methods

We compare PACE against four state-of-the-art inference-time reasoning methods: DPTS (Ding et al., 2025), SSDP (Kim et al., 2025), FETCH (Wang et al., 2025b), and SWAP (Xiong et al., 2025). These baselines cover MCTS-style tree search, semantic pruning, verifier-regularized search, and structured planning. Appendix G describes the baselines and implementation details.

5.2 Performance Metrics

Accuracy. The fraction of problems solved correctly. Following standard practice in this line of work (Ding et al., 2025; Kim et al., 2025; Wang et al., 2023), we select the final answer via majority vote over all candidates produced for a problem.

Average Token Usage. The mean number of generated tokens per problem, used as a measure of computational efficiency. We report this alongside

Table 1: Accuracy percentage on GSM8K and Math500 under fixed token budgets. Token counts denote actual generated output tokens per problem, reported in thousands. For SWAP, token counts include both generator output tokens and discriminator output tokens.

Budget	Method	GSM8K				Math500			
		Llama-3.2-3B		Llama-3.1-8B		Llama-3.2-3B		Llama-3.1-8B	
		Acc. $\uparrow$	Tok. $\downarrow$	Acc. $\uparrow$	Tok. $\downarrow$	Acc. $\uparrow$	Tok. $\downarrow$	Acc. $\uparrow$	Tok. $\downarrow$
3k	DPTS	28.35	3.1k	43.75	3.4k	23.20	3.9k	21.20	3.9k
	SSDP	23.73	1.1k	56.56	1.2k	40.20	2.0k	46.80	2.2k
	FETCH	52.54	2.9k	54.21	2.9k	13.00	2.9k	12.40	2.9k
	SWAP	64.67	7.7k	85.90	5.4k	31.60	6.1k	48.60	7.4k
	PACE	82.03	3.0k	86.13	3.0k	43.80	3.0k	43.40	3.0k
	PACE-ES	79.23	2.1k	85.60	2.6k	40.80	2.8k	41.60	2.9k
5k	DPTS	35.10	3.9k	57.39	4.4k	35.80	5.9k	33.80	6.0k
	SSDP	22.97	1.1k	57.09	1.2k	41.80	2.3k	48.80	2.7k
	FETCH	59.97	4.9k	63.99	4.9k	12.60	4.9k	16.80	4.9k
	SWAP	74.00	17.7k	87.11	16.3k	30.00	10.8k	51.00	13.1k
	PACE	84.53	5.0k	88.02	5.0k	47.80	5.0k	48.60	5.0k
	PACE-ES	78.77	2.4k	85.60	3.0k	51.80	4.0k	47.80	4.3k
8k	DPTS	37.60	4.3k	61.26	4.9k	43.40	7.6k	43.00	7.6k
	SSDP	22.37	1.1k	56.33	1.2k	43.20	2.8k	50.60	3.2k
	FETCH	67.85	7.7k	73.21	8.0k	16.80	7.9k	19.00	7.9k
	SWAP	74.60	28.6k	86.95	28.2k	33.40	20.0k	54.20	24.3k
	PACE	83.02	8.0k	87.57	8.1k	50.00	8.0k	53.40	8.0k
	PACE-ES	82.26	2.5k	87.04	3.2k	49.00	5.3k	50.60	5.8k
10k	DPTS	39.80	4.4k	62.24	5.0k	45.60	8.6k	47.20	8.5k
	SSDP	22.37	1.1k	56.79	1.2k	43.00	3.0k	50.20	3.3k
	FETCH	71.04	9.6k	76.27	9.8k	16.00	9.9k	20.40	9.9k
	SWAP	74.60	35.6k	86.80	35.4k	31.80	26.8k	52.60	33.3k
	PACE	84.15	10.0k	87.34	10.0k	49.40	9.9k	52.20	10.0k
	PACE-ES	81.43	2.6k	86.66	3.3k	51.80	6.0k	52.80	6.6k

accuracy to distinguish methods that achieve accuracy through better search from those that achieve it through additional compute.

Coverage. The fraction of problems for which at least one correct solution appears among the first N generated candidates (Shi et al., 2025; Lyu et al., 2025; Walder and Karkhanis, 2025). Coverage characterizes the exploration capability of a search framework. We report it as a function of N to characterize how candidate diversity translates into reachable accuracy.

6 Performance Results

Table 1 reports accuracy and token usage for PACE and four baselines across two benchmarks, two model scales, and four budget regimes. Figure 5 reports coverage as a function of the number of generated candidates N . Additional evaluations in Appendix H compare PACE against simple inference-time sampling baselines and evaluate the compari-

son with Qwen-PRM-7B to test whether the conclusions persist under a different scorer.

6.1 Accuracy and Token Usage

Comparison with Tree-Search Baselines. As observed in Table 1, DPTS exhibits high initial overhead inherent to MCTS where several rollouts are needed before value estimates become discriminative (Dam et al., 2024), which hurts performance at tight budgets (most visibly at 3k on Math500), and leaves larger budgets underutilized due to its early stopping mechanism. In contrast, SSDP performs best among the tree-search baselines under tight constraints on Math500, where aggressive semantic-similarity pruning maximizes the utility of limited tokens. However, this strategy does not scale with larger budgets: the same aggressive pruning produces an empty search frontier, leaving the controller with no valid nodes to expand, a possible limitation the authors acknowledge. On GSM8K

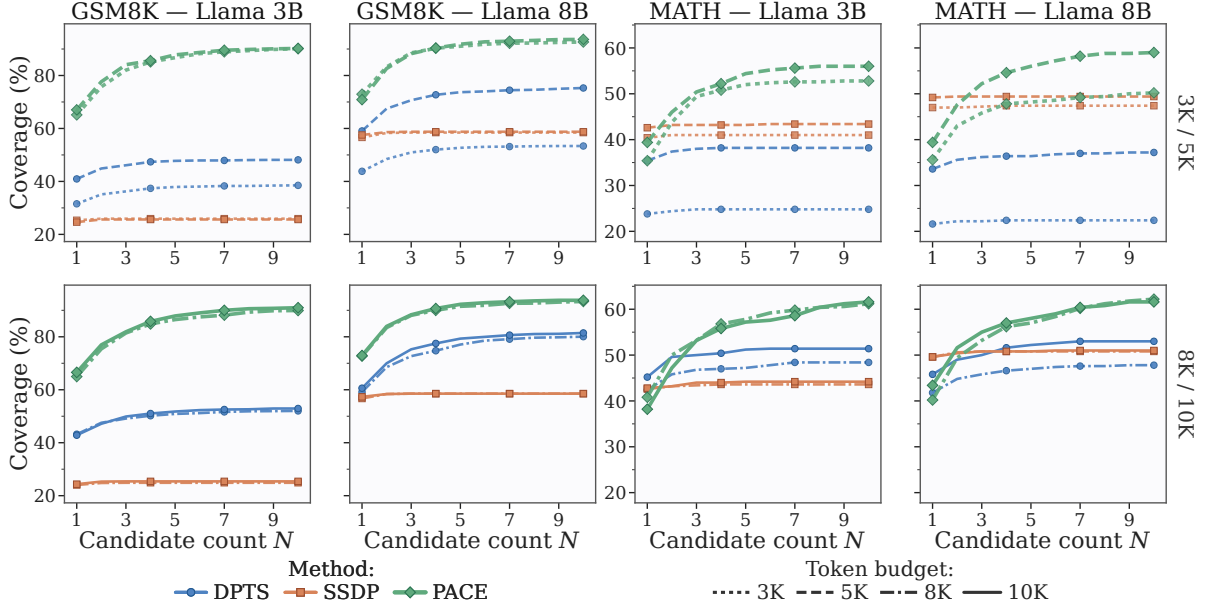


Figure 5: Coverage as a function of candidate count N , across benchmarks and model scales and four token budgets.

with Llama-3.2-3B at the 3k budget, 1,308 of 1,319 problem instances terminated early because the frontier became empty. FETCH approaches its nominal budget more fully than SSDP and produces a more selective merging policy. However, on Math500, FETCH appears more sensitive to the imposed budget. The original FETCH results report Math500 accuracy around 40% using roughly 23k generated tokens. Under our fixed-budget setting, where runs are stopped at 3k–10k tokens, this larger-compute behavior does not fully materialize on Math500. This suggests that FETCH improves search quality when sufficient compute is available, but is less elastic on more complex problems under tighter resource limits.

Behavior of PACE. PACE’s gains over the baselines are larger on GSM8K than on Math500 because GSM8K problems are typically shorter, spanning only 2–8 reasoning steps (Cobbe et al., 2021). As a result, an initial solution is often found early, leaving more of the budget for exploring distinct regions of the search space. This also explains the saturation effect across GSM8K budgets: once a high-quality solution is identified under lower constraints, additional budget yields smaller marginal accuracy gains. To examine this accuracy–efficiency tradeoff, we evaluate PACE-ES, a variant that adapts an early-stopping rule. PACE-ES achieves competitive accuracy on GSM8K using substantially fewer tokens.

SWAP exceeds PACE in three Math500/Llama-

8B settings (48.60 vs. 43.40 at 3k; 51.00 vs. 48.60 at 5k; 54.20 vs. 53.40 at 8k). In each case, however, it uses roughly 2.5–3 $\times$ the generated-token count. SWAP additionally relies on benchmark-specific adapter training for both its policy and discriminator components, whereas PACE is applied without modification across benchmarks.

6.2 Coverage Analysis

Accuracy under majority vote depends both on whether the candidate set contains a correct trajectory and on whether the voting rule selects it. Coverage, reported in Figure 5, isolates the former.

DPTS’s coverage flattens beyond $N \approx 5$ at every budget, consistent with its early stopping rule that once a sufficiently high-reward terminal is found, further rollouts are pursued only if they exceed it. SSDP’s coverage is nearly flat in N and varies little across budgets, the coverage-side counterpart to its frontier-depletion behavior in the token-usage data: semantic merging removes candidates faster than the controller produces new ones, so additional budget yields few additional unique trajectories.

PACE’s gain over the other baselines is largest on GSM8K, consistent with the accuracy results. The shape of PACE’s own coverage curve differs by benchmark. On GSM8K, it saturates quickly, since most problems produce multiple candidates even at tight budgets. On Math500, longer reasoning chains mean that under tight budgets many problems never produce more than one or two candidates and therefore cannot contribute to

Table 2: Ablation of controller components on Math500/ Llama-3.2-3B at 3k, 5k, 8k token budget. SC: seeking control; RC: refinement control; AB: adaptive branching.

Budget	Configuration	Acc. $\uparrow$	Cov@5 $\uparrow$
3k	Base	33.80	34.80
	+SC	39.80	42.80
	+SC+RC	41.40	45.40
	+SC+RC+AB	43.80	49.60
5k	Base	36.60	38.80
	+SC+RC	40.80	43.60
	+SC+RC+AB	47.80	51.40
8k	Base	41.20	43.00
	+SC+RC	46.60	49.00
	+SC+RC+AB	50.00	55.00

coverage@ N at higher N . As the budget grows, more problems clear the per-problem candidate count needed to enter the coverage@ N statistics, so the curve continues to climb with both N and budget. The growing coverage of PACE at increasing N suggests that its gains are at least partly attributable to candidate diversity rather than to selection alone.

6.3 Isolating the Effect of Search Policy

To isolate the contribution of the search policy from the quality of the scorer, we compare PACE with a step-level MCTS controller equipped with the same PRM. MCTS already balances exploitation and exploration through UCT. In this control, we additionally initialize each node’s value using its PRM score, while keeping the generator, token budgets, and final aggregation unchanged. If the gains of PACE were primarily due to the scoring signal, this PRM-initialized MCTS should obtain similar performance. Instead, on Math500 with Llama-3.2-3B at the 8k budget, MCTS reaches 21.2% accuracy compared with 50.0% for PACE. Under a finite budget, MCTS frequently fails to reach a complete trajectory: 252 of 500 problems at 3k and 151 of 500 at 8k terminate without producing a candidate. This indicates that the difference is not only in how partial states are scored, but in how the search allocates its limited budget toward reaching and refining complete candidates.

6.4 Ablation Study

To quantify the contribution of each component, we ablate the controller components on Math500 with Llama-3.2-3B at different token budgets. Table 2 reports accuracy and coverage at $N = 5$ for the base method and for each successive addition: seeking control (SC), refinement control (RC), and adaptive branching (AB). At 3k, we report the contribution of each component separately. SC is the largest single contributor, improving accuracy by 6.0 points and coverage by 8.0 points. RC adds a further 1.6 / 2.6 points, while AB adds another 2.4 / 4.2 points. At 5k and 8k, we report SC and RC jointly; together they improve accuracy by 4.2 and 5.4 points, respectively, while AB adds a further 7.0 and 3.4 points. Adaptive branching improves both metrics, consistent with its role of reducing branching when sibling continuations are redundant and allocating additional expansion to diverse continuations. Coverage improves more than accuracy at every budget, showing that the components help the search find correct candidates more often, even when the final selected answer is not correct.

7 Conclusion

In this paper, we highlight the limitations of fixed-policy search space exploration approaches adopted in modern Tree-of-Thought (ToT) frameworks that lead to suboptimal reasoning results under limited token budget due to premature over-expansion of shallow nodes and exhausting tokens and/or localization of exploration only on a few dominant branches of the tree. To this end, we propose PACE (Phase-Aware Controller for Exploration), which adaptively switches between discover and diversify phases across competing branches. We propose depth- and candidate(s)-aware seeking and refining regimes, and spectral diversity-aware adaptive branching to guide the reasoning search space in a principled fashion. Evaluations across different models, datasets, and token budgets demonstrate that PACE outperforms recent tree-search baselines in most settings, with gains of up to 56% in relative accuracy over the strongest tree-search baseline. Comparison with PRM-initialized MCTS further demonstrates the importance of an effective search policy, while evaluations with Qwen-PRM-7B show that PACE remains effective under a different scorer. Overall, our results highlight the importance of adapting budget allocation to the current state of the search.

Limitations

Scope of Evaluation. The empirical findings here are obtained on two mathematical reasoning benchmarks with two open-weights model scales. Mathematical reasoning is a useful proving ground because it provides verifiable terminal states and well-studied PRMs, but it does not represent the full space of multi-step reasoning tasks. The regime-conditioning principle behind our strategy is not specific to mathematics, but the PRM used to score partial trajectories was trained on mathematical reasoning data; extending the analysis to other domains may require a domain-appropriate PRM.

Single-Run Reporting. We report the results of each configuration from a single run rather than the mean and standard deviation across multiple runs. Run-to-run variance in our setting comes primarily from generator sampling, and small differences on specific cells should be interpreted with this in mind.

When Discover-Then-Diversify May Not Apply. Our discover-then-diversify rule assumes that reaching *one complete answer early* is useful, since under budget exhaustion a complete but sub-optimal candidate is better than none. This holds for our math reasoning benchmarks, where a correct trajectory is reachable at moderate depth, and the final answer appears at a terminal step. This assumption may not hold on tasks where the answer is not reached by moving toward the first complete trajectory, and where an early commitment must instead be undone by backtracking. For example, *planning-intensive logical reasoning* tasks (Hao et al., 2025), where committing early to one path can lead to a dead end, may instead benefit from a search policy that delays deep commitment and keeps multiple options open.

Compute Cost is Measured in Tokens, Not Wall-Clock Time. Following standard practice in the tree-search-for-reasoning literature (Miyamoto et al., 2026; Schwartz et al., 2020; Wang et al., 2025b; Xiong et al., 2025; Wang et al., 2025a), we report cost in generated tokens. The cost comparisons should be read as comparisons of generated tokens, not of end-to-end inference latency.

Acknowledgments

This work is supported in part by the U.S. Department of Energy (DOE), Office of Science, Office

of Advanced Scientific Computing Research under contract DEAC02-06CH11357/0F-60169 and the National Science Foundation (NSF) under award no. 2411386/2411387, 2514056. AI tools were used in a limited capacity to improve clarity and were fully verified by the authors.

References

- Lukas Aichberger, Kajetan Schweighofer, Mykyta Ielanskyi, and Sepp Hochreiter. 2025. [Improving uncertainty estimation through semantically diverse language generation](#). In *ICLR: The Thirteenth International Conference on Learning Representations*.
- Argonne Leadership Computing Facility. 2026. [Polaris](https://www.alcf.anl.gov/polaris). <https://www.alcf.anl.gov/polaris>. Accessed: May 1, 2026.
- Maciej Besta, Nils Blach, Ales Kubicek, Robert Gertenberger, Michał Podstawski, et al. 2024. [Graph of thoughts: solving elaborate problems with large language models](#). In *AAAI: Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence*.
- Guillaume M. J.-B. Chaslot, Mark H. M. Winands, Jos W. H. M. Uiterwijk, H. Jaap van den Herik, and Bruno Bouzy. 2007. [Progressive strategies for monte-carlo tree search](#). In *Information Sciences*.
- Giannis Chatziveroglou. 2025. [A*-decoding: Token-efficient inference scaling](#).
- Tristan Cinquin, Geoff Pleiss, and Agustinus Kristiadi. 2025. [Limits of PRM-guided tree search for mathematical reasoning with LLMs](#). In *NeurIPS: The 5th Workshop on Mathematical Reasoning and AI*.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, et al. 2021. [Training verifiers to solve math word problems](#). *arXiv preprint*.
- Tuan Dam, Odalric-Ambrym Maillard, and Emilie Kaufmann. 2024. [Power mean estimation in stochastic monte-carlo tree search](#). In *Proceedings of the Fortieth Conference on Uncertainty in Artificial Intelligence*. PMLR.
- Yifu Ding, Wentao Jiang, Shunyu Liu, Yongcheng Jing, Jinyang Guo, et al. 2025. [Dynamic parallel tree search for efficient llm reasoning](#). In *ACL: Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, et al. 2024. [The llama 3 herd of models](#).
- Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason E Weston, and Yuandong Tian. 2025. [Training large language models to reason in a continuous latent space](#). In *Second Conference on Language Modeling*.

- Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. [Measuring mathematical problem solving with the MATH dataset](#). In *NeurIPS: Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.
- Joongho Kim, Xirui Huang, Zarreen Reza, Gabriel Grand, Kevin Zhu, and Ryan Lagasse. 2025. [Chopping trees: Semantic similarity based dynamic pruning for tree-of-thought reasoning](#). In *NeurIPS 2025 Workshop on Efficient Reasoning*.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. [Efficient memory management for large language model serving with pagedattention](#). In *Proceedings of the 29th Symposium on Operating Systems Principles*. Association for Computing Machinery.
- Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. [Let’s verify step by step](#). In *ICLR: The Twelfth International Conference on Learning Representations*.
- Zhicun Lyu, Xinye Li, Zheng Xie, and Ming Li. 2025. [Top pass: improve code generation by pass@k-maximized code ranking](#). *Front. Comput. Sci.*
- Atkia Mahila, Avinash Maurya, M. Mustafa Rafique, and Bogdan Nicolae. 2026. [Beyond fixed budgets: Characterizing the inelasticity and limitations of tree-of-thought reasoning strategies](#). In *HPDC: 35th International Symposium on High-Performance Parallel and Distributed Computing*.
- Sora Miyamoto, Daisuke Oba, and Naoaki Okazaki. 2026. [Aligning tree-search policies with fixed token budgets in test-time scaling of LLMs](#). In *ICML: Forty-third International Conference on Machine Learning*.
- Olivier Roy and Martin Vetterli. 2007. [The effective rank: A measure of effective dimensionality](#). In *EU-SIPCO: Proceedings of the European Signal Processing Conference*.
- Roy Schwartz, Gabriel Stanovsky, Swabha Swayamdipta, Jesse Dodge, and Noah A. Smith. 2020. [The right tool for the job: Matching model and instance complexities](#). In *ACL: 58th Annual Meeting of the Association for Computational Linguistics*.
- H. S. Seung, M. Oppen, and H. Sompolsky. 1992. [Query by committee](#). In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*.
- Weijie Shi, Yue Cui, Yaguang Wu, Jingzhi Fang, Shibo Zhang, Mengze Li, Sirui Han, Jia Zhu, Jiajie Xu, and Xiaofang Zhou. 2025. [Semantic-guided diverse decoding for large language model](#). In *NeurIPS: The Thirty-ninth Annual Conference on Neural Information Processing Systems*.
- Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2025. [Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning](#). In *ICLR: The Thirteenth International Conference on Learning Representations*.
- Zhongxiang Sun, Qipeng Wang, Weijie Yu, Xiaoxue Zang, Kai Zheng, Jun Xu, Xiao Zhang, Yang Song, and Han Li. 2025. [Rearter: Retrieval-augmented reasoning with trustworthy process rewarding](#). In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*.
- Christian Walder and Deep Tejas Karkhanis. 2025. [Pass@k policy optimization: Solving harder reinforcement learning problems](#). In *NeurIPS: The Thirty-ninth Annual Conference on Neural Information Processing Systems*.
- Ziyu Wan, Xidong Feng, Muning Wen, Stephen Marcus McAleer, Ying Wen, et al. 2024. [Alphazero-like tree-search can guide large language model decoding and training](#). In *ICML: International Conference on Machine Learning*.
- Ante Wang, Linfeng Song, Ye Tian, Baolin Peng, Dian Yu, et al. 2025a. [Litesearch: Efficient tree search with dynamic exploration budget for math reasoning](#). In *AAAI: Proceedings of the AAAI Conference on Artificial Intelligence*.
- Ante Wang, Linfeng Song, Ye Tian, Dian Yu, Haitao Mi, Xiangyu Duan, et al. 2025b. [Don’t get lost in the trees: Streamlining llm reasoning by overcoming tree search exploration pitfalls](#). In *ACL: Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*.
- Junlin Wang, Siddhartha Jain, Dejiao Zhang, Baishakhi Ray, Varun Kumar, and Ben Athiwaratkun. 2024a. [Reasoning in token economies: Budget-aware evaluation of LLM reasoning strategies](#). In *EMNLP: Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.
- Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2024b. [Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations](#). In *ACL: 62nd Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.
- Rui Wang, Zeming Wei, Guanzhang Yue, and Meng Sun. 2025c. [Revisiting hallucination detection with effective rank-based uncertainty](#). *Preprint*, arXiv:2510.08389.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. [Self-consistency improves chain of thought reasoning in language models](#). In *ICLR: The Eleventh International Conference on Learning Representations*.

- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. [Chain-of-thought prompting elicits reasoning in large language models](#). In *NeurIPS: 36th International Conference on Neural Information Processing Systems*.
- Yuxi Xie, Anirudh Goyal, Wenyue Zheng, Min-Yen Kan, Timothy P Lillicrap, Kenji Kawaguchi, and Michael Shieh. 2024. [Monte carlo tree search boosts reasoning via iterative preference learning](#). In *The First Workshop on System-2 Reasoning at Scale, NeurIPS'24*.
- Siheng Xiong, Ali Payani, Yuan Yang, and Faramarz Fekri. 2025. [Deliberate reasoning in language models as structure-aware planning with an accurate world model](#). In *ACL: 63rd Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.
- Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, et al. 2023. [Tree of thoughts: deliberate problem solving with large language models](#). In *NeurIPS: 37th International Conference on Neural Information Processing Systems*.

A Baseline PRM-Guided Best-First Controller

Algorithm 1 presents the PRM-guided best-first search controller used as the non-adaptive baseline throughout our analysis. The controller maintains a global frontier of partial reasoning states, ranks frontier nodes by their PRM scores, expands the top-ranked nodes, and adds completed trajectories to the candidate set.

Algorithm 1 Baseline PRM-Guided Search

```

1: Input: prompt  $x$ , LLM generator  $G$ , PRM
   scorer  $R$ 
2: Parameters: tree width  $W$ , branch factor  $B$ ,
   generated-token budget  $T$ 
3: Initialize frontier  $\mathcal{F}_0 \leftarrow \{\text{root}(x)\}$ 
4: Initialize candidate set  $\mathcal{C} \leftarrow \emptyset$ 
5:  $t \leftarrow 0$ 
6: while budget remains and  $\mathcal{F}_t \neq \emptyset$  do
7:   Select top- $W$  nodes  $P_t \subseteq \mathcal{F}_t$  ranked by
   PRM score
8:    $\mathcal{F}_t \leftarrow \mathcal{F}_t \setminus P_t$ 
9:   for all  $p \in P_t$  do
10:     $U \leftarrow G(p, B)$ 
11:    for all  $u \in U$  do
12:      Evaluate  $R(u)$ 
13:      if  $u$  contains a valid answer then
14:        Add  $u$  to  $\mathcal{C}$ 
15:      else
16:        Insert  $u$  into  $\mathcal{F}_t$ 
17:      end if
18:    end for
19:  end for
20:   $t \leftarrow t + 1$ 
21: end while
22: return  $\text{Aggregate}(\mathcal{C})$ 

```

B Depth Saturation Statistics

Section 3.1 shows that the baseline controller concentrates its expansion mass within a shallow band of depths. Table 3 reports the dynamics of this saturation as the token budget is consumed. We bin expansion steps into fixed checkpoints and report, at each checkpoint, the cumulative share of the token budget consumed, the mean depth of expanded nodes, and the 95th-percentile maximum depth reached across problems.

The qualitative pattern is that mean expanded depth saturates well before the token budget is exhausted. The 95th-percentile maximum depth

Table 3: Depth saturation of baseline. Mean expanded depth saturates near 3.2 by expansion step 100, while the 95th-percentile maximum depth reaches near 8 over the same interval, indicating that the budget is exhausted before the controller meaningfully deepens its expansion mass.

Expansion step	Budget consumed	Mean exp. depth	95th pct. max depth
50	55.7	2.8	6
100	93.2	3.2	7
150	98.3	3.3	8
200	99.3	3.3	8

shows that the controller can reach deeper regions of the tree, but these deeper excursions account for only a small fraction of the expansion mass. Thus, the baseline does not redistribute its budget toward deeper regions as search progresses; instead, it continues to spend most of its budget in the shallow band identified in Section 3.1.

C Post-Candidate Locality: Protocol and Full Distributions

Section 3.2 examines how the baseline allocates search after a candidate has already been discovered. Here we describe the controlled handoff and the dominant-subtree measure used in that analysis.

Protocol. We measure post-candidate behavior on a subset of Math500 problems using a controlled handoff: once a candidate-seeking procedure finds a first candidate, we transfer the search state to the baseline PRM-guided controller and log its subsequent expansions under the remaining token budget. The handoff only provides a comparable post-candidate starting point; all locality measurements are computed from the baseline’s behavior afterward.

Dominant-Subtree Share. For an anchor depth d^* , we partition the explored tree into disjoint subtrees rooted at depth- d^* ancestors. Let $\{T_1, T_2, \dots, T_k\}$ denote these subtrees, and let $\mathcal{C}$ be the set of candidates generated after handoff. We define the dominant-subtree share as:

$$\rho(d^*) = \max_i \frac{|\mathcal{C} \cap T_i|}{|\mathcal{C}|} \quad (4)$$

A value of $\rho(d^*) = 1$ means that every post-handoff candidate originates from the same depth- d^* subtree. Lower values indicate that candidates are distributed across multiple subtrees. We evaluate $d^* \in \{2, 3, 4\}$ to measure concentration at

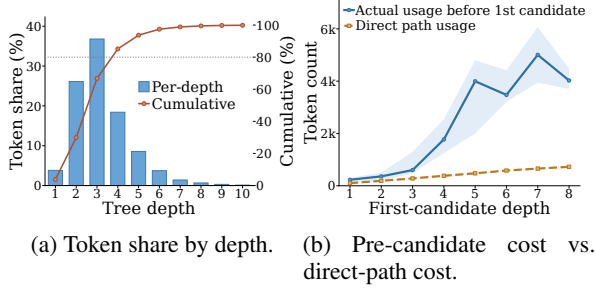


Figure 6: **Observation 1 under Qwen-PRM.** Budget allocation of the baseline. Left: most tokens are spent at shallow depths. Right: pre-candidate cost grows much faster than direct-path cost as first-candidate depth increases.

progressively finer levels of the search tree. The resulting distributions are shown in Figure 3.

D Observations under a Different Process Reward Model

Section 3 characterizes the baseline controller using Math-Shepherd-PRM-7B as the step-level scorer. To test whether the observed behaviors depend on that particular scorer, we repeat the same analysis with Qwen-PRM-7B. The generator (Llama-3.2-3B), benchmark (Math500), token budget (8k), and measurement protocol are unchanged; only the PRM is replaced.

Observation 1: Budget Concentrates at Shallow Depths. Figure 6a reports the share of tokens spent at each depth. The cumulative token share reaches 85.1% by depth four and 93.7% by depth five, showing the same concentration of budget at shallow depths. Figure 6b compares the tokens consumed before the first candidate is discovered with the cost of the corresponding root-to-candidate path. The gap again increases with candidate depth. At depth five, the controller consumes about 4.0k tokens before discovering the first candidate, while the direct path to that candidate costs only about 471 tokens. Table 4 provides the complementary expansion-step view: by expansion step 100, 93.4% of the budget has been consumed while mean expanded depth is only 3.3. The 95th-percentile maximum depth reaches 8, indicating that deeper nodes are reached but receive relatively little of the total expansion budget.

Observation 2: Candidate Sets Concentrate in Few Subtrees. Following the same protocol as Appendix C, Table 5 reports the dominant-subtree share at anchor depths $d^* \in \{2, 3, 4\}$. At $d^* = 2$,

Table 4: Depth saturation of the baseline under Qwen-PRM-7B on Math500 / Llama-3.2-3B at an 8k token budget.

Expansion step	Budget consumed (%)	Mean exp. depth	95th pct. max depth
50	55.5	2.9	7
100	93.4	3.3	8
150	98.8	3.4	8
200	99.7	3.4	8

Table 5: Dominant-subtree concentration under Qwen-PRM-7B on Math500 / Llama-3.2-3B.

Anchor depth d^*	Problems with $\rho \geq 0.75$
2	78.6%
3	53.8%
4	39.0%

78.6% of eligible problems have at least 75% of their post-candidate solutions concentrated in one subtree, and 65.9% have $\rho = 1$. The latter is consistent with the 64% observed with Math-Shepherd, showing that the same post-candidate concentration pattern persists under a different PRM. Concentration decreases as the anchor depth increases, but remains substantial at $d^* = 3$ and $d^* = 4$.

Overall, both observations persist when the step-level scorer is changed from Math-Shepherd-PRM-7B to Qwen-PRM-7B, indicating that the behaviors identified in Section 3 are not specific to a single PRM.

E Adaptive Response Specification

This appendix specifies the functional form and feedback law for each of the three responses introduced in §4.1 and §4.2:

Depth Response α_t (Candidate-Seeking Regime). At each iteration, we compute the spread of PRM scores on the current frontier as the normalized difference between the maximum and median:

$$\alpha_t = \max_{v \in \mathcal{F}_t} \text{PRM}(v) - \text{median}_{v \in \mathcal{F}_t} \text{PRM}(v) \quad (5)$$

When frontier scores cluster tightly, the max-minus-median is small and α_t takes its floor value; when frontier scores spread, α_t grows toward the ceiling.

Locality Response λ_t (Candidate-Refining Regime). The locality response responds to whether recent selections continue to come from the candidate’s subtree. After each selection step

in the refining regime, we examine the structural overlap of the selected nodes with the most recent candidate c . If recent selections are still drawn predominantly from c 's subtree, we increase λ_{t+1} to push subsequent selections further away.

After each selection step, we compute the mean structural overlap $L(v, c)$ from Eq. 2 between the selected nodes v and the most recent candidate c . If this mean exceeds the overlap threshold $\kappa_{\text{ov}} = 0.8$, then recent selections are still being drawn predominantly from the subtree rooted near c . Thus, λ_t rises gradually when the search becomes too localized, and relaxes slowly once the controller begins selecting structurally different nodes. We report a sensitivity study for κ_{ov} in Appendix I.

Depth-Alignment Response μ_t (Candidate-Refining Regime). We define

$$D(v) = \tanh\left(\frac{\text{depth}(v) - d_{\text{ref}}}{s(d_{\text{ref}})}\right) \quad (6)$$

where d_{ref} is the depth of the most recent candidate. The scale $s(d_{\text{ref}})$ widens the tolerance band as candidates appear deeper: when the first candidate is shallow, the controller is more sensitive to depth differences; when it is deep, the band is correspondingly wider. The $\tanh$ bounds the contribution to $[-1, 1]$ so that the depth term never overrides the PRM signal entirely.

The coefficient μ_t is updated based on the depth of recently selected nodes relative to a pivot $p = d_{\text{ref}} - \Delta$, where $\Delta = \max(1, \lceil \ln(d_{\text{ref}} + 1) \rceil)$. The logarithmic band is intentionally conservative: it is narrow for problems with shallow candidates, and grows for deeper candidate problems. If recent selections fall below p , μ_{t+1} is increased to anchor more strongly at the candidate depth; if they fall within the band $[p, d_{\text{ref}}]$, μ_{t+1} is held constant.

F Adaptive Branching Policy

To regulate branching, we estimate the semantic diversity among an initial probe set of child continuations. Intuitively, if the probe samples are highly similar, further expansion is unlikely to provide substantially new reasoning trajectories. If they are diverse, additional branching (Chaslot et al., 2007) is more justified.

Let $\{y_i\}_{i=1}^m$ denote the probe continuations generated from parent node p , and let $z_i \in \mathbb{R}^d$ denote their normalized embedding vectors. We form the cosine Gram matrix

$$G_{ij}(p) = z_i^\top z_j. \quad (7)$$

Let $\{\lambda_k\}_{k=1}^m$ be the eigenvalues of $G(p)$, clipped to be nonnegative and normalized as:

$$\pi_k(p) = \frac{\lambda_k}{\sum_j \lambda_j} \quad (8)$$

We then define the spectral entropy

$$H(p) = - \sum_{k=1}^m \pi_k(p) \log \pi_k(p), \quad (9)$$

and the corresponding mode count

$$M(p) = \exp(H(p)) \quad (10)$$

$\mathcal{M}(p) \in [1, m]$ behaves as an *effective number of semantic modes* present among the sibling continuations. This construction follows the notion of entropy-based effective rank as a real-valued extension of matrix rank (Roy and Vetterli, 2007). Recent LLM uncertainty work also uses effective-rank-based spectral structure as a meaningful uncertainty signal for model generations and hidden representations (Wang et al., 2025c). Operationally, low $\mathcal{M}(p)$ indicates that sibling samples collapse to a small set of near-equivalent meanings (high local confidence / redundancy), while high $\mathcal{M}(p)$ indicates meaningful disagreement across samples. This aligns with classic disagreement-based uncertainty principles (e.g., query-by-committee) (Seung et al., 1992) and with recent LLM uncertainty work emphasizing semantic-space variability over token-level heuristics (Aichberger et al., 2025).

The controller uses this signal in two ways. First, it determines whether the parent should be expanded beyond the probe stage. Second, it controls redundancy-aware retention of generated children before reinsertion into the frontier. Algorithm 2 summarizes the complete procedure of our method.

G Compared Approaches

DPTS (Ding et al., 2025). A Monte Carlo Tree Search variant adapted to LLM reasoning. DPTS uses a Search-and-Transition module that gates compute allocation: during an initial free-exploration phase the controller pursues rollouts broadly, and once $t^* = 5$ terminal solutions have been produced, a strict best-solution-aware stopping rule applies and new rollouts are pursued only if the selected node's reward strictly exceeds the best known terminal reward. As with all MCTS-style methods, DPTS depends on a sampling volume sufficient to stabilize its value estimates; under tight budgets, this warm-up cost becomes visible (Mahila et al., 2026).

Algorithm 2 PACE: Regime-conditioned Search

```
1: Input: prompt  $x$ , LLM generator  $G$ , PRM  
   scorer  $R$   
2: Parameters: generated-token budget  $T$   
3: Initialize frontier  $\mathcal{F}_0 \leftarrow \{\text{root}(x)\}$   
4: Initialize candidate set  $\mathcal{C} \leftarrow \emptyset$   
5:  $t \leftarrow 0$   
6: while budget remains and  $\mathcal{F}_t \neq \emptyset$  do  
7:   if  $\mathcal{C} = \emptyset$  then  
8:     Compute candidate-seeking scores using Eq. (1)  
9:   else  
10:    Compute candidate-refining scores using Eq. (3)  
11:  end if  
12:  Select top node  $P_t \subseteq \mathcal{F}_t$   
13:  for all  $p \in P_t$  do  
14:     $U \leftarrow \text{AdaptiveBranching}(p, B)$   
15:    for all  $u \in U$  do  
16:      Evaluate PRM( $u$ )  
17:      if  $u$  contains a valid answer then  
18:        Add  $u$  to  $\mathcal{C}$   
19:      else  
20:        Insert  $u$  into  $\mathcal{F}_t$   
21:      end if  
22:    end for  
23:  end for  
24:  if  $\mathcal{C} \neq \emptyset$  then  
25:    Update  $\lambda_{t+1}$  and  $\mu_{t+1}$   
26:  end if  
27:   $t \leftarrow t + 1$   
28: end while  
29: return  $\text{Aggregate}(\mathcal{C})$ 
```

Implementation. We use the authors’ reported default settings: tree width 4, maximum depth 16.

SSDP (Kim et al., 2025). A semantic-pruning extension of DPTS that mitigates branch explosion through online merging. Each newly generated node is embedded and compared by cosine similarity to its siblings; pairs exceeding a fixed threshold τ are merged irrevocably, and the unexplored children of the merged-away node are discarded. SSDP is efficient under tight budgets but exhibits frontier depletion when τ is permissive relative to the geometry of the embedding space, since semantically distinct paths can be collapsed before a viable solution is found. *Implementation.* We use the authors’ reported default settings, including the merge threshold $\tau = 0.75$.

FETCH (Wang et al., 2025b). A plug-and-play tree-search framework that targets two failure modes identified by its authors: *over-exploration* (redundant states with semantically equivalent content) and *under-exploration* (high variance in verifier scoring causing trajectory switching). FETCH merges semantically similar states via agglomerative clustering of fine-tuned SimCSE embeddings and trains its verifier with temporal-difference λ -returns and ensembling to reduce score variance.

Implementation. We instantiate FETCH on its MCTS backbone, since the original paper reports MCTS+FETCH as the strongest tree-search variant among their evaluated search backbones. The released FETCH code ships task-specific components for GSM8K (a GSM8K-finetuned Llama-3-8B policy, two verifier checkpoints value-A and value-B, and a GSM8K SimCSE merge model). To avoid giving FETCH a task-specific policy advantage unlike other baselines, we only replace its finetuned policy models with the same vanilla instruct policy models used everywhere else in this paper. For GSM8K, we use the authors’ released GSM8K verifier ensemble and released GSM8K SimCSE merge model. For MATH, the authors mention using an ensemble of two Math-Shepherd-style PRMs in place of its GSM8K verifier setup. The public repository does not release two FETCH-specific MATH PRM checkpoints, so we instantiate a Math-Shepherd-style ensemble using peiyi9979/math-shepherd-mistral-7b-prm and OpenRLHF/Mistral-7b-PRM-Math-Shepherd. The public release likewise does not appear to include a MATH-specific SimCSE merge checkpoint; rather than reuse the GSM8K merge model out-of-domain, we train a MATH-domain merge encoder using the authors’ provided embedding-training script.

SWAP (Xiong et al., 2025). A structure-aware planning framework. SWAP follows a different approach from verifier-guided tree search. Instead of building a search tree over partial solutions, it keeps an explicit entailment graph that represents the current reasoning state. At each step, a policy model proposes possible graph expansions, while a world model predicts how the graph should be updated. A discriminator then ranks the alternative updates, and a diversity-based sampling rule helps the method explore different reasoning directions. We include SWAP as a structured-reasoning

Table 6: Accuracy (%) of simple inference-time baselines and PACE using Math-Shepherd.

Method	3k	5k	8k	10k
GSM8K – Llama-3.2-3B				
Weighted BoN	60.05	67.17	72.25	73.92
Self-Consistency	59.29	68.69	72.25	74.68
PACE	82.03	84.53	83.02	84.15
GSM8K – Llama-3.1-8B				
Weighted BoN	56.03	59.14	60.80	60.80
Self-Consistency	52.62	51.78	55.42	56.03
PACE	86.13	88.02	87.57	87.34
Math500 – Llama-3.2-3B				
Weighted BoN	36.20	40.60	44.00	47.00
Self-Consistency	35.40	38.80	43.80	45.00
PACE	43.80	47.80	50.00	49.40
Math500 – Llama-3.1-8B				
Weighted BoN	27.80	31.40	32.00	35.80
Self-Consistency	25.00	25.80	28.00	27.80
PACE	43.40	48.60	53.40	52.20

baseline rather than as a direct tree-search baseline. This allows us to compare our method against a recent planning-based controller that uses explicit symbolic structure.

Implementation. For the 8B setting, we use the released SWAP-v2 LoRA adapters for GSM8K and MATH. For the 3B setting, the authors do not provide released SWAP adapters. We therefore train benchmark-specific generator and discriminator LoRA adapters using the authors’ training scripts and released supervision data. For fixed-budget evaluation, we modify the SWAP evaluation loop to generate candidates incrementally and stop once the generator completion-token budget is reached. The discriminator is then applied only to the candidates generated within this budget. Therefore, the token budget in SWAP is enforced against the policy model’s generation; the discriminator’s re-ranking calls are not counted against it. Reported completion tokens are the sum of generator output tokens and discriminator output tokens.

H Additional Results

H.1 Additional Inference-Time Baselines

Table 6 reports additional comparison results against two simple sampling baselines that do not construct a search tree. *Weighted BoN* samples independent chain-of-thought trajectories and selects the final answer using PRM-weighted voting, while *Self-Consistency* (Wang et al., 2023) uses majority voting over the same samples. Both baselines use

the same generator and token budgets as PACE.

PACE outperforms both sampling baselines across all budgets, datasets, and model sizes. The gap is particularly large on GSM8K, while the differences are smaller but consistent on Math500. Increasing the budget improves the sampling baselines, but does not close the gap to PACE. These results show that the gains are not explained simply by generating more independent reasoning trajectories within the same budget.

H.2 Results with a Different Process Reward Model

To test whether the observed gains depend on the choice of process reward model, we evaluate the same search methods with Qwen-PRM-7B. The results are shown in Table 7. The main trends remain similar under the alternative scorer. On GSM8K, PACE maintains a large advantage over DPTS and SSDP across all budgets and both model sizes. Results on Math500 are closer, with SSDP demonstrating comparable results in almost all configurations.

The coverage results in Figure 7 show the same distinction in search behavior. SSDP’s coverage changes little as more candidates are considered, whereas PACE continues to gain coverage with increasing N . This trend holds across budgets and model sizes, indicating that PACE continues to produce a broader set of candidate solutions under Qwen-PRM.

Taken together, these results indicate that the gains reported in Section 6 follow from how the controller allocates its budget rather than from the particular PRM used to rank partial states.

I Sensitivity Analysis

The coefficients α_t , λ_t , and μ_t are adaptive quantities that evolve with the search state. Each is initialized once and then updated at every iteration from the current search state, as described in Appendix E. Thus, after initialization, their values adapt to the behavior of the controller rather than requiring separate tuning throughout the search.

To verify that these coefficients respond dynamically in practice, we track their values on a representative Math500/Llama-3.2-3B run with an 8k token budget. Both λ_t and μ_t traverse their full range from 0 to 1, while α_t varies from 0.10 to 0.63. Each coefficient moves in both directions during search; for example, λ_t increases 2,805 times

Table 7: Accuracy percentage on GSM8K and Math500 under fixed token budgets, with Qwen-PRM-7B as the step-level scorer in place of Math-Shepherd-PRM-7B. All other components: generator, budgets, aggregation, and baseline implementations, are identical to Table 1.

Budget	Method	Qwen-PRM-7B							
		GSM8K				Math500			
		Llama-3.2-3B		Llama-3.1-8B		Llama-3.2-3B		Llama-3.1-8B	
		Acc. ↑	Tok. ↓	Acc. ↑	Tok. ↓	Acc. ↑	Tok. ↓	Acc. ↑	Tok. ↓
3k	DPTS	47.31	3.0k	41.39	3.3k	27.60	3.8k	24.20	3.9k
	SSDP	53.15	1.0k	57.01	1.1k	48.00	2.0k	48.80	2.1k
	PACE	78.17	3.0k	83.55	3.0k	47.40	3.0k	49.80	3.0k
	PACE-ES	76.88	2.0k	83.32	2.2k	49.20	2.5k	49.80	2.7k
5k	DPTS	55.50	3.6k	54.28	4.1k	37.60	5.7k	35.80	5.7k
	SSDP	54.44	1.1k	55.80	1.1k	50.40	2.5k	52.40	2.6k
	PACE	78.92	5.0k	84.15	5.0k	52.00	5.0k	53.40	5.0k
	PACE-ES	77.03	2.5k	82.56	2.6k	49.20	3.6k	54.40	3.8k
8k	DPTS	57.39	3.8k	57.16	4.4k	46.00	7.0k	46.60	6.8k
	SSDP	53.37	1.1k	56.94	1.2k	53.00	2.7k	55.00	3.0k
	PACE	80.44	8.0k	85.06	8.0k	51.40	8.0k	55.60	8.0k
	PACE-ES	79.98	3.0k	84.91	3.0k	51.40	4.8k	55.00	5.2k
10k	DPTS	57.39	3.9k	57.32	4.5k	50.40	7.7k	50.60	7.6k
	SSDP	52.69	1.1k	55.88	1.2k	53.00	2.9k	54.80	3.2k
	PACE	80.06	10.0k	84.53	10.0k	57.00	10.0k	55.40	10.0k
	PACE-ES	78.32	3.2k	85.82	3.2k	52.00	5.6k	56.80	5.9k

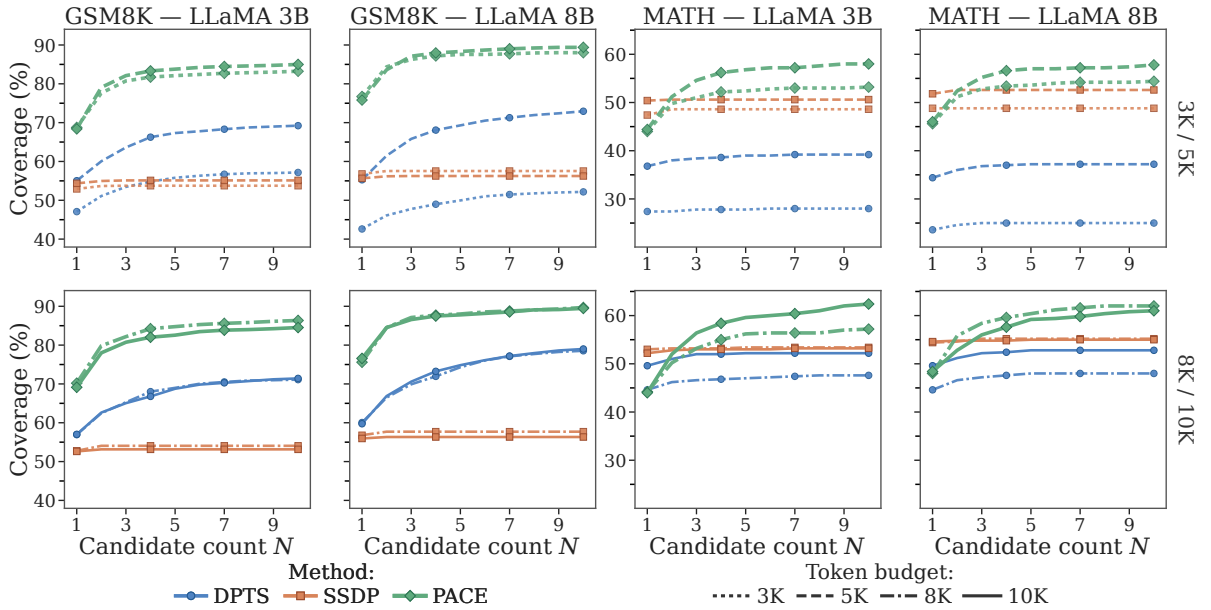


Figure 7: Coverage as a function of candidate count N with Qwen-PRM-7B as the step-level scorer. Top row: 3k and 5k budgets. Bottom row: 8k and 10k.

Table 8: Accuracy of PACE on Math500 using Llama-3.2-3B at the 3k token budget as a function of the overlap threshold κ_{OV} . The value used throughout the main results is $\kappa_{OV} = 0.8$.

κ_{OV}	0.3	0.5	0.6	0.7	0.8	0.9
Accuracy	43.2	43.0	41.4	42.8	43.8	43.2

and decreases 6,097 times.

The overlap threshold κ_{OV} , in contrast, is a fixed parameter that determines when the locality response λ_t increases versus when it relaxes. We therefore evaluate its sensitivity on Math500 with Llama-3.2-3B at the 3k token budget, varying κ_{OV} across six values. Table 8 reports the resulting accuracy.

Accuracy is relatively stable across the tested overlap thresholds. The best result is obtained at $\kappa_{OV} = 0.8$, which is the value used in the main experiments. Most thresholds remain close to this value: $\kappa_{OV} \in \{0.3, 0.5, 0.7, 0.9\}$ are within 1.0 accuracy point of the best result.

J Reproduction Notes

We follow the dataset formatting, final-answer extraction, and answer-normalization pipeline of DPTS (Ding et al., 2025). All generator calls are served through vLLM (Kwon et al., 2023) using shared decoding parameters, with per-node maximum generation length set to 100 as adopted from DPTS; generated-token budgets count output tokens only, and prompt tokens are excluded. We use Math-Shepherd-PRM-7B (Wang et al., 2024b) as the step-level scorer, matching the PRM-guided setting used by recent tree-search methods such as DPTS, SSDP, and FETCH.

Prompt Format and Worked Example. All dataset items are converted to the prompt template below before search. We use the same template for

every method at a given model scale.

Prompt template

Please reason step by step, and ensure that the final answer includes the correct unit. Output format MUST be exactly one line at the end. Put your final answer on its own line between tags like this:
`<<FINAL>> <answer> <</FINAL>>`
For example: `<<FINAL>> 90 <</FINAL>>`
Do not output anything after that final line.
Problem: {question}
Solution:

A worked Math500 example:

Example question and node-wise trajectory

Question. $\overline{BC}$ is parallel to the segment through A , and $AB = BC$. What is the number of degrees represented by x ?

Selected trajectory.

Node 1. ## Step 1: First, let's recall the properties of parallel lines and transversals in geometry. When a transversal intersects two parallel lines, the corresponding angles are congruent, and the interior angles on the same side of the transversal are supplementary.
Step 2: In this problem, $\overline{BC}$ is parallel to the segment through A . This means that $\angle BAC$ and $\angle ABC$ are corresponding angles, and their

Node 2. measures are equal.

Step 3: Since we're given that $AB = BC$, $\triangle ABC$ is an isosceles triangle. In an isosceles triangle, the base angles are congruent, so $\angle ABC = \angle ACB$.

Step 4: We're also given that $\angle BAC = 124^\circ$. Since the sum of the angles in a triangle is 180° , we can

Node 3. find the measures of $\angle ABC$ and $\angle ACB$ using the formula $x + x + 124 = 180$.

Step 5: Simplifying the equation, we get $2x + 124 = 180$. Solving for x , we have $2x = 180 - 124 = 56$, which gives $x = \frac{56}{2} = 28$.

Step 6: Therefore, the measure

Node 4. of angle x is 28° .

`<<FINAL>> 28 <</FINAL>>`