Backtracking with Linear-in-Depth Search-Space Growth: Width-Limited Tree Search for Large Language Models

Jiacheng Xu[1,2] **Fuxiang Zhang**[1,2] **Chaojie Wang**[2] **Bo An**[1]
[1]Nanyang Technological University, Singapore [2]Skywork AI
jiacheng005@e.ntu.edu.sg, boan@ntu.edu.sg

Abstract

Inference-time search offers a principled way to improve the reliability of large language models (LLMs) by exploring and scoring intermediate trajectories. However, widely used search paradigms exhibit complementary failure modes: sequence search scales linearly but commits early and cannot recover from mistaken prefixes, whereas tree search can backtrack but often becomes impractical for deep search due to the exponential growth of the search space with depth. These limitations become increasingly pronounced in long-chain-of-thought responses. We propose *Width-Limited Tree Search* (WLTS), which reconciles these strengths by preserving the tree-search structure that enables backtracking while using a dynamic expandability criterion to enforce a strict per-layer width constraint. This design keeps generated search-tree growth linear in depth and sustains verifier supervision across the full reasoning process. WLTS further supports two optional deployment components: prefix deletion for diversity and agreement-based early stopping to reduce computation on tasks with canonicalizable outputs. Across mathematical, coding, and logical reasoning benchmarks, and under both classifier and generative verifier settings, WLTS achieves higher accuracy and better inference-budget efficiency than well-established sequence- and tree-search baselines.

1 Introduction

Large language models (LLMs) have demonstrated remarkable capabilities and have been widely adopted across diverse applications [17, 5]. However, the quality of responses generated in a single-pass, serial manner is often unstable, which has led to increasing attention on techniques for improving inference-time performance, especially for multi-step reasoning tasks [27, 20, 30]. Among these, search algorithms stand out by simultaneously exploring multiple candidate intermediate trajectories and selecting the most promising ones using a reward model or verifier [15, 19, 34, 31].

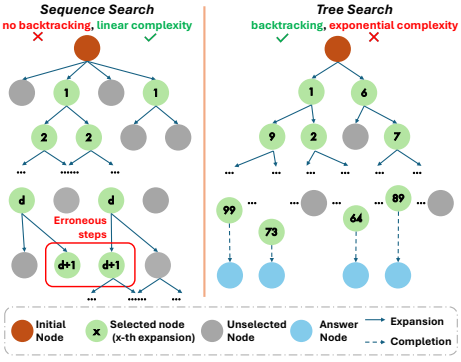


Figure 1: Limitations of sequence and tree search. The left may get stuck in erroneous intermediate nodes and cannot backtrack. The right experiences exponential search-space growth with depth, so limited resources restrict it to shallow searches.

From the perspective of backtracking, search algorithms for LLMs can be broadly divided into sequence search and tree search [12, 28]. As shown in Figure 1, sequence search, such as best-of-N and beam search [20], does not allow backtracking: once the candidate set is saturated with

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

erroneous steps, the model can only continue reasoning from those mistakes. In contrast, tree search, including Monte Carlo Tree Search (MCTS) [2] and recent LLM-oriented variants [32, 4], explores multiple continuations from each intermediate state and can return to an earlier decision point when later evidence reveals an unpromising branch. This backtracking improves fault tolerance by allowing the search to abandon erroneous trajectories and try alternative continuations. However, without structural control, the number of generated search-tree nodes can grow as $O(b^d)$ with branching factor b and reasoning depth d. Under limited inference budgets, this exponential growth often causes tree search to over-explore shallow layers and rely on single-pass completion for the remaining reasoning steps [24, 8]. This phenomenon becomes more pronounced as reasoning horizons, model sizes, and context lengths increase [23].

Because backtracking offers stronger fault tolerance and a higher performance ceiling, much recent work on LLM inference has focused on tree-search methods that allocate inference-time computation more effectively [6]. Existing approaches are often guided by two signals: *value-guided* allocation uses reward models or verifiers to score partial trajectories and prune nodes or prioritize expansions [22, 24, 23], while *uncertainty-aware* allocation uses uncertainty, variance, or expected information gain to balance exploration and exploitation [3, 7]. These directions are often combined, but they mainly reallocate expansion budgets without inherently enforcing a strict per-layer capacity limit. Consequently, generated search-tree nodes can still grow exponentially with depth, concentrating finite budgets in shallow layers and weakening verifier supervision over deeper reasoning steps [6, 8, 23].

To address this structural bottleneck, we propose *Width-Limited Tree Search* (WLTS), a tree-search algorithm that preserves backtracking while constraining generated search-tree nodes to grow linearly with reasoning depth. WLTS retains the standard selection–expansion–evaluation–backpropagation loop, but replaces unconstrained branching with a strict per-layer width cap W enforced by a dynamic expandability criterion. This criterion permits expansion only at or beyond the deepest saturated layer, forcing search budget to propagate along depth rather than accumulate in shallow branches. Among eligible nodes, WLTS expands the highest-priority node by one reasoning step, sustaining verifier supervision over deeper reasoning while retaining the ability to revise earlier decisions through backtracking. WLTS also includes two optional deployment components: prefix deletion to reuse freed width for diversity, and agreement-based early stopping to reduce computation on tasks with canonicalizable outputs.

We summarize our contributions as follows:

- To the best of our knowledge, WLTS is the first tree-search-style method for LLM inference that supports backtracking while guaranteeing linear-in-depth growth of generated search-tree nodes.
- We provide two configurable mechanisms that are naturally compatible with WLTS, supporting practical tradeoffs between accuracy and efficiency.
- Through extensive experiments on challenging mathematical, coding, and logical reasoning benchmarks, under both classifier and generative reward model settings, we show that WLTS achieves higher accuracy and better inference-budget efficiency than well-established baselines.

2 Background

We briefly review the reward models and search algorithms used as points of comparison for WLTS. Let x denote the question prompt and $y = (y^{(1)}, \ldots, y^{(d)})$ the trajectory segmented into d *steps*, each $y^{(t)}$ being an LLM-generated text span, delimited or length-limited. A partial trajectory $y^{(1:t)}$ defines a node at depth t, with the proposer $p_\theta(y^{(t+1)} \mid x, y^{(1:t)})$ generating next-step continuations and the verifier $r_\phi(x, y^{(1:t)})$ scoring partial trajectories to guide search.

2.1 Reward Model

Search algorithms rely on a scoring function to compare partial trajectories and select final outputs [9]. Given a prompt x and partial trajectory $y^{(1:t)}$, a reward model provides a scalar signal $r_\phi(x, y^{(1:t)})$, either from a trained classifier [16, 25] or from a prompted generative verifier [18, 34, 19, 33]. For the latter, we prompt a verifier LLM to emit a discrete judgment, e.g., *Is the existing reasoning correct*

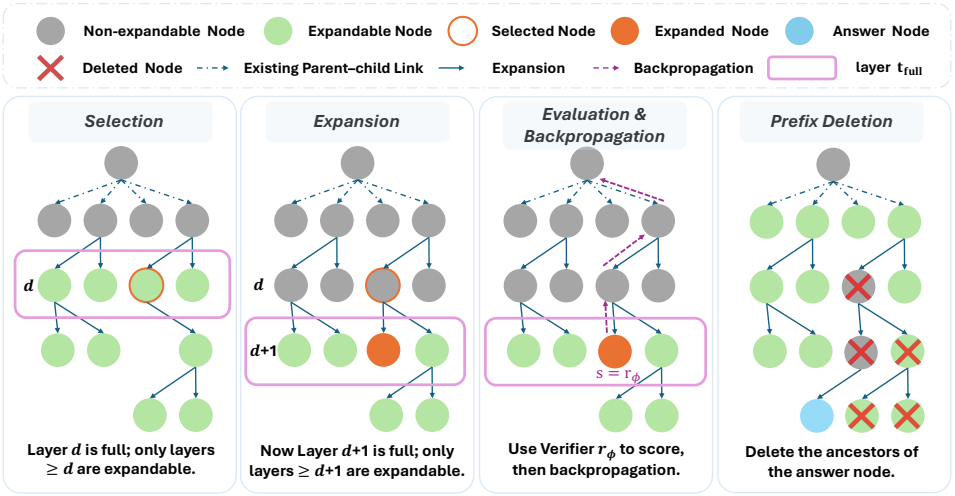


Figure 2: Overview of Width-Limited Tree Search (WLTS). WLTS maintains layer-wise search nodes under a strict per-layer width cap, repeatedly selects an eligible node by priority score, expands it by one reasoning step, evaluates it with a verifier, and backpropagates the score. When a terminal trajectory is found, WLTS may apply optional deletion to free width capacity and diversify exploration.

(Yes/No)?, and convert the returned token probability into a scalar score. Let $\hat{z} \in \{\texttt{Yes}, \texttt{No}\}$ denote the generated judgment token and let $\ell_\phi(\hat{z} \mid x, y^{(1:t)})$ be its log probability; we use the signed mapping

$$r_\phi(x, y^{(1:t)}) = \begin{cases} +\exp(\ell_\phi(\hat{z} \mid x, y^{(1:t)})), & \hat{z} = \texttt{Yes}, \\ -\exp(\ell_\phi(\hat{z} \mid x, y^{(1:t)})), & \hat{z} = \texttt{No}. \end{cases}$$

By varying prompt granularity, the same generative reward model can act as an outcome verifier or as a process/progress verifier that provides dense per-step feedback [15].

2.2 Search Algorithms

Beam Search. Beam search is a standard width-limited *sequence* search method, commonly used as a baseline for LLM inference [21]. It maintains a beam $\mathcal{B}_t$ of up to W partial trajectories at depth t, expands each by b next-step continuations, scores the resulting candidates, and keeps the top W as $\mathcal{B}_{t+1}$. This yields a predictable $O(Wbd)$ candidate evaluation cost at depth d, but pruning is irreversible: with noisy step-level scores or correlated proposals, the beam can collapse onto an incorrect path and force subsequent search to extend it. WLTS inherits the benefit of explicit width control, but keeps a tree structure so that earlier prefixes can still be revisited through backtracking.

Monte Carlo Tree Search. Monte Carlo Tree Search (MCTS) incrementally grows a search tree by repeatedly running simulations and reallocating computation toward promising branches [2, 24]. In the LLM setting, nodes are partial trajectories and edges are step-level continuations; each simulation follows selection, expansion, evaluation, and backpropagation. A common selection rule is UCT [10]: given a node $y^{(1:t)}$ with expanded children $\mathrm{C}(y^{(1:t)})$, visit count $n(y^{(1:t)})$, and empirical mean return $v(y^{(1:t)})$, UCT selects the next child by

$$y^{(1:t+1)*} = \operatorname*{arg\,max}_{y' \in \mathrm{C}(y^{(1:t)})} \left(v(y') + c_{\mathrm{uct}} \sqrt{\frac{\log n(y^{(1:t)})}{n(y') + 1}} \right),$$

where $c_{\mathrm{uct}} > 0$ controls exploration.

Unlike beam search, MCTS enables *backtracking*: as evidence accumulates, it can revisit earlier prefixes and reallocate budget to alternative branches. However, without structural constraints, generated search-tree nodes can scale as $O(b^d)$ with branching factor b and depth d. Under finite budgets, this growth often confines exploration to shallow layers. WLTS retains this backtracking capability while enforcing per-layer width control to keep generated search-tree growth linear in depth.

3 Width-Limited Tree Search

3.1 Motivation

We aim to retain the ability to backtrack, which is the key advantage of tree search for LLM reasoning, without allowing the generated search tree to grow exponentially with depth. Figure 2 provides an overview of WLTS. The central design principle is to enforce a strict *per-layer* width cap, forcing search resources to propagate along depth rather than accumulate in shallow branches. Intuitively, WLTS can revisit and extend earlier reasoning states whenever later evidence suggests that a branch is unpromising, as long as it satisfies the width constraint.

3.2 Search Procedure

Each node in the search tree corresponds to a partial trajectory $y^{(1:t)}$. For each depth t, let $\mathcal{L}_t$ collect the nodes kept at that depth. WLTS enforces the strict per-layer width constraint

$$|\mathcal{L}_t| \le W, \quad \forall t,$$

For each node $y^{(1:t)}$, WLTS stores a visit count $n(y^{(1:t)})$ and a value estimate $v(y^{(1:t)})$, the empirical mean of verifier rewards observed from expansions through this node. The search repeats selection–expansion–evaluation–backpropagation until it collects N terminal trajectories or meets a stopping criterion, and returns the best terminal trajectory by verifier score.

Initialization. The root node is the empty trajectory ε (depth 0), whose associated state includes the prompt x. Initially, $\mathcal{L}_0 = \{\varepsilon\}$ and $\mathcal{L}_t = \emptyset$ for all $t \ge 1$. We initialize $n(\varepsilon) = 1$ and $v(\varepsilon) = 0$ so that the UCT bonus $\log n(\varepsilon)$ is well-defined. The search hyperparameters are the width cap W, depth limit $D_{\max}$, expansion budget $B_{\max}$, terminal-candidate budget N, and deletion depth h_{del} used by the optional deletion mechanism in Section 3.3.

Selection. WLTS uses a dynamic *expandability* rule to enforce the per-layer width cap. We call a layer $t \ge 1$ *saturated* if $|\mathcal{L}_t| = W$, and define t_{full} as the deepest saturated layer; if no layer is saturated, we set $t_{\mathrm{full}} = 0$:

$$t_{\mathrm{full}} \triangleq \max\Big(\{t \ge 1 : \ |\mathcal{L}_t| = W\} \cup \{0\}\Big).$$

Expanding a depth-t node creates exactly one child at depth $t+1$. By definition of t_{full}, every layer strictly deeper than t_{full} is *not* saturated. Therefore, restricting expansion to depths $t \ge t_{\mathrm{full}}$ guarantees that any child inserted into $\mathcal{L}_{t+1}$ lands in a layer with remaining width capacity. We define the expandable set as

$$\mathcal{E} \triangleq \bigcup_{t \ge t_{\mathrm{full}}} \mathcal{L}_t.$$

Among expandable nodes, WLTS selects using an exploration–exploitation score,

$$y^\star \ = \ \arg\max_{y \in \mathcal{E}} \left(v(y) + c_{\mathrm{uct}} \sqrt{\frac{\log n(\varepsilon)}{n(y)+1}} \right),$$

where $c_{\mathrm{uct}} \ge 0$ controls exploration and $n(\varepsilon)$ is the root visit count, which tracks the total number of expansions so far. Setting $c_{\mathrm{uct}} = 0$ recovers greedy best-first selection; ties can be broken by preferring deeper nodes.

Expansion. Given the selected node $y^{(1:t)}$, WLTS samples a next-step continuation from the proposer, $y^{(t+1)} \sim p_\theta(\cdot \mid x, y^{(1:t)})$, and forms the child node $y^{(1:t+1)}$. In our implementation, a step is terminated either by a designated step delimiter or by a fixed token budget.

Evaluation and Backpropagation. The new node is scored by the verifier reward model, $s \triangleq r_\phi(x, y^{(1:t+1)})$ (Section 2.1). We set $n(y^{(1:t+1)}) \leftarrow 1$ and $v(y^{(1:t+1)}) \leftarrow s$. The score s is then propagated upward along the path to the root. For each ancestor node u on this path, we update its visit count and running-average value estimate by

$$n(u) \leftarrow n(u) + 1, \qquad v(u) \leftarrow v(u) + \frac{s - v(u)}{n(u)}.$$

Algorithm 1 Width-Limited Tree Search (WLTS)

Require: prompt x, proposer p_θ, verifier r_ϕ, width cap W, depth limit $D_{\max}$, terminal target N, expansion budget $B_{\max}$, exploration coefficient c_{uct}, deletion depth h_{del}

Initialize $\mathcal{L}_0 \leftarrow \{\varepsilon\}$, $\mathcal{L}_t \leftarrow \emptyset$ for $t \geq 1$, and $\mathcal{T} \leftarrow \emptyset$
Initialize $n(\varepsilon) \leftarrow 1$ and $v(\varepsilon) \leftarrow 0$
for $k = 1$ to $B_{\max}$ **do**
  $t_{\text{full}} \leftarrow \max\left(\{t \geq 1 : |\mathcal{L}_t| = W\} \cup \{0\}\right)$
  $\mathcal{E} \leftarrow \bigcup_{t \geq t_{\text{full}}} \mathcal{L}_t$
  Select $y^{(1:t)} \leftarrow \arg\max_{y \in \mathcal{E}} \left(v(y) + c_{\text{uct}} \sqrt{\frac{\log n(\varepsilon)}{n(y)+1}} \right)$
  Sample $y^{(t+1)} \sim p_\theta(\cdot \mid x, y^{(1:t)})$ and form $y^{(1:t+1)}$
  Evaluate $s \leftarrow r_\phi(x, y^{(1:t+1)})$; set $n(y^{(1:t+1)}) \leftarrow 1$ and $v(y^{(1:t+1)}) \leftarrow s$
  Backpropagate s to update $(n(\cdot), v(\cdot))$ on the path to the root
  if $y^{(1:t+1)}$ is terminal or $t + 1 = D_{\max}$ **then**
    $\mathcal{T} \leftarrow \mathcal{T} \cup \{y^{(1:t+1)}\}$; apply prefix deletion (Section 3.3)
    if $|\mathcal{T}| \geq N$ **then**
      break
    end if
  else
    Insert $y^{(1:t+1)}$ into $\mathcal{L}_{t+1}$
  end if
end for
return $\arg\max_{y \in \mathcal{T}} r_\phi(x, y)$

If the child trajectory is terminal or reaches $D_{\max}$, it is added to the terminal candidate set $\mathcal{T}$. Otherwise, the child node is inserted into the corresponding layer $\mathcal{L}_{t+1}$. Finding a terminal trajectory may also trigger optional deletion as described below.

3.3 Diversity Enhancement

Vanilla search algorithms can over-invest in a single high-scoring branch and produce many minor variants. WLTS encourages diversity via controllable *prefix deletion*. Fix a deletion depth $h_{\text{del}} \geq 1$. For any kept node $y^{(1:t)}$, define its truncated prefix

$$\pi_{h_{\text{del}}}\left(y^{(1:t)}\right) \triangleq y^{(1:\min\{h_{\text{del}},t\})}.$$

This induces *families* over kept nodes: two nodes belong to the same family if they share the same $\pi_{h_{\text{del}}}$. When a terminal trajectory $y^{(1:T)}$ is found, let $p = \pi_{h_{\text{del}}}(y^{(1:T)})$ and delete the remaining kept nodes in that family:

$$\mathcal{L}_t \leftarrow \{u \in \mathcal{L}_t : \pi_{h_{\text{del}}}(u) \neq p\}, \quad \forall t.$$

This frees width to explore alternative prefixes; setting $h_{\text{del}} = 1$ recovers root-level family deletion (grouping by the first generated step).

3.4 Early Stop

In contrast to sequence search methods, which advance multiple hypotheses in parallel, and classic tree search, which typically completes a batch of frontier nodes only after the search budget is exhausted, WLTS generates fully completed trajectories *sequentially* in the order of their priority scores. This output ordering enables a natural, optional early-stopping mechanism based on agreement among terminal candidates.

For tasks whose final answers admit a canonical representation, we maintain a running frequency table over the answers extracted from completed trajectories. After WLTS finishes each trajectory, we canonicalize its predicted answer and update its count. If any answer reaches a predefined occurrence threshold M, we stop the search early and return the highest-scoring trajectory among those producing that majority answer. This strategy reduces unnecessary exploration when WLTS rapidly converges to a consistent solution, while preserving robustness when candidate answers diverge.

Table 1: Benchmark results across various datasets. Acc.(%) and Tok.(k) report accuracy and average generator token usage (in thousands), respectively. **Average** columns report macro-averaged accuracy and tokens across the four datasets in each block; parentheses next to accuracy indicate absolute improvements over Best-of-N (in percentage points), and parentheses next to tokens indicate token usage relative to Best-of-N (= 100%). WLTS is highlighted for readability.

Classifier RM	MATH500		OlympiadBench		AMC23		MBPP		Average	
	Acc.(%)	Tok.(k)	Acc.(%)	Tok.(k)	Acc.(%)	Tok.(k)	Acc.(%)	Tok.(k)	Acc.(%)	Tok.(k)
Llama-3.1-8B	46.00	–	16.59	–	30.00	–	65.90	–	39.62	–
+ Best-of-N	66.20	12.68	28.30	23.16	50.00	20.24	72.50	4.42	54.25	15.13 (100%)
+ Beam Search	70.20	9.87	33.33	14.47	52.50	13.08	71.70	4.10	56.93 (+2.68)	10.38 (69%)
+ MCTS	66.40	68.93	30.67	86.37	55.00	91.65	72.50	13.39	56.14 (+1.89)	65.09 (430%)
+ Q*	69.80	57.15	32.74	60.39	55.00	62.46	73.10	14.22	57.66 (+3.41)	48.56 (321%)
+ ABMCTS-A	66.20	38.16	30.39	42.48	52.50	41.34	70.90	18.39	55.00 (+0.75)	35.09 (232%)
+ **WLTS**	**72.40**	16.98	**36.30**	30.54	**65.00**	30.05	**75.90**	7.57	**62.40** (+8.15)	21.29 (141%)
Generative RM	AIME24		AIME25		ZebraLogic		LiveCodeBench		Average	
	Acc.(%)	Tok.(k)	Acc.(%)	Tok.(k)	Acc.(%)	Tok.(k)	Acc.(%)	Tok.(k)	Acc.(%)	Tok.(k)
Qwen3-8B	22.40	–	21.07	–	32.90	–	23.21	–	24.90	–
+ Best-of-N	29.33	60.39	22.67	50.95	41.10	36.42	29.52	9.41	30.66	39.29 (100%)
+ Beam Search	32.67	72.75	27.33	55.16	45.80	52.60	31.93	16.53	34.43 (+3.77)	49.26 (125%)
+ MCTS	34.67	145.41	28.67	121.93	46.50	107.82	30.12	67.00	34.99 (+4.33)	110.54 (281%)
+ Q*	35.33	87.41	27.33	74.87	46.70	54.70	31.33	61.75	35.17 (+4.51)	69.68 (177%)
+ ABMCTS-A	32.67	61.69	**29.33**	58.88	40.80	53.66	29.52	50.76	33.08 (+2.42)	56.25 (143%)
+ **WLTS**	**42.67**	42.91	**29.33**	38.21	**48.40**	36.41	**33.13**	14.16	**38.38** (+7.72)	32.92 (84%)
Qwen3-14B	28.27	–	25.07	–	44.90	–	27.71	–	31.49	–
+ Best-of-N	40.67	56.16	32.00	46.06	50.60	34.22	31.93	9.09	38.80	36.38 (100%)
+ Beam Search	40.00	76.74	33.33	62.52	55.10	33.84	34.34	18.55	40.70 (+1.90)	47.91 (132%)
+ MCTS	42.00	155.00	33.33	152.75	54.40	112.89	34.94	40.98	41.17 (+2.37)	115.41 (317%)
+ Q*	38.00	57.28	33.66	51.57	51.40	62.38	**38.55**	43.42	40.40 (+1.60)	53.66 (147%)
+ ABMCTS-A	40.67	60.20	31.33	59.93	52.20	53.61	37.95	42.53	40.54 (+1.74)	54.07 (149%)
+ **WLTS**	**46.00**	32.58	**35.33**	33.70	**63.60**	37.69	37.99	14.64	**45.73** (+6.93)	29.65 (82%)

3.5 Complexity Analysis

Let G denote the number of generated step-level search-tree nodes for a single input instance, from initialization until WLTS returns an output. For unconstrained tree search, the search space to depth d contains $O(b^d)$ nodes under branching factor b, so exhaustive exploration requires $G = O(b^d)$ generated nodes. WLTS instead uses the expandability rule to enforce $|\mathcal{L}_t| \leq W$ at every depth, preventing search resources from accumulating in shallow layers. Without prefix deletion, this width invariant gives $G \leq WD_{\max}+N$. With prefix deletion, let C be the number of deletion operations. Since each terminal trajectory can trigger at most one deletion operation, $C \leq N$, and applying the same width bound within each deletion interval gives $G \leq (C+1)WD_{\max}+N \leq (N+1)WD_{\max}+N = O(NWD_{\max})$. Thus, for fixed search-budget parameters, WLTS retains backtracking while avoiding exponential growth in generated search-tree nodes. Appendix A provides the detailed accounting.

4 Experiments

4.1 Experiment Setup

To more comprehensively evaluate WLTS, we conduct experiments using multiple representative model configurations. For the classifier reward model, we use Llama-3.1-8B-Instruct as the base model and Skywork-o1-Open-PRM-Qwen-2.5-7B as the reward model [16]. For the generative reward model, we use Qwen3-8B and Qwen3-14B as the base models, and use them as their own verifiers, as they possess the ability to make effective judgments. Given the strength of the base models, we select challenging reasoning tasks that cover both mathematical and coding domains to comprehensively evaluate WLTS. For Llama-3.1-8B-Instruct, we use math problems from MATH500, Olympiad Bench, AMC23, and coding problems from MBPP. For Qwen3-8B and Qwen3-14B, we use math problems from AIME24 and AIME25, coding problems from LiveCodeBench, and the logical-reasoning benchmark ZebraLogic. For sequence search, we use best-of-N and beam search as representative baselines. For tree search, we include MCTS, as well as Q* [24] and ABMCTS-A [8], two recent efficiency-oriented tree-search variants.

Table 2: Component-wise ablation of WLTS on Qwen3-14B. Each cell reports accuracy (%) / generated tokens (k) per problem. PD = prefix deletion; ES = agreement-based early stopping. The Beam Search and ABMCTS-A rows reproduce Table 1 for reference.

Method	AIME24	AIME25	LiveCodeBench	ZebraLogic	Average
Beam Search	40.00 / 76.74	33.33 / 62.52	34.34 / 18.55	55.10 / 33.84	40.70 / 47.91
ABMCTS-A	40.67 / 60.20	31.33 / 59.93	<u>37.95</u> / 42.53	52.20 / 53.61	40.54 / 54.07
Core WLTS	41.33 / 27.28	32.33 / 25.88	37.35 / 4.09	55.80 / 26.51	41.70 / 20.94
Core WLTS + ES	42.67 / 19.72	31.33 / 18.84	37.35 / 4.09	<u>57.50</u> / 11.85	42.21 / 13.63
Core WLTS + PD	**47.33** / 50.01	**36.67** / 52.45	**37.99** / 14.64	<u>57.50</u> / 54.97	<u>44.87</u> / 43.02
WLTS	<u>46.00</u> / 32.58	<u>35.33</u> / 33.70	**37.99** / 14.64	**63.60** / 37.69	**45.73** / 29.65

4.2 Main Results

Evaluation protocol. Following prior test-time scaling evaluations [20, 33, 14], we use a *candidate-count-matched protocol*, where each method targets up to $N{=}16$ complete candidate answers per problem. Best-of-N and Beam Search directly decode N complete trajectories, whereas MCTS, Q*, and ABMCTS-A search over step-level prefixes under fixed expansion budgets and, when fewer than N full trajectories are produced, complete selected prefixes with a single forward pass following the common budget-limited tree-search completion protocol [24]. WLTS uses the same candidate target and may terminate earlier if the early stopping condition is met. Because matched candidate counts do not imply matched token costs, we report total generated tokens for the full search procedure, including intermediate branches, discarded candidates, and final completions; Appendix F provides a token-budget-matched comparison.

Results. Table 1 shows that inference-time search improves over single-pass decoding across all model and verifier settings, and step-level search generally outperforms trajectory-level Best-of-N. However, existing tree-search baselines offer no compelling advantage over Beam Search once their substantially higher token usage is considered. The key reason is that, in long-CoT reasoning, finite search budgets are often spent disproportionately on shallow regions, leaving deeper reasoning steps without sustained verifier supervision and forcing completion from the current best prefix, which degrades final answer quality. Although Q* and ABMCTS-A reduce token usage by limiting the number of children expanded at each step, under fixed expansion budgets they do not yield a substantial accuracy advantage over standard MCTS. In contrast, WLTS is designed for this long-CoT setting: it inherits the linearly constrained search space of sequence search, distributing budget more evenly across the full reasoning process, while preserving the recovery structure of tree search so that backpropagation can return to more promising prefixes after errors. This recovery behavior is empirically important: with Qwen3-14B on AIME24 and AIME25, 59 of 60 solved AIME24 instances and all 51 solved AIME25 instances are solved only after at least one backtracking event; moreover, 10 and 9 solved instances, respectively, are solved only after WLTS first reaches an incorrect terminal answer and then recovers through backtracking, providing direct evidence of recovery behavior unavailable to beam-style search.

4.3 Components Ablation

WLTS combines three elements: (i) the core width-limited backtracking structure, (ii) prefix deletion (PD) for diversity, and (iii) agreement-based early stopping (ES). To disentangle their contributions, we ablate the optional components on Qwen3-14B and report accuracy and generated-token usage across AIME24, AIME25, LiveCodeBench, and ZebraLogic. We also include Beam Search and ABMCTS-A from Table 1 as reference baselines, since they provide the strongest accuracy–token trade-offs among non-WLTS methods in this setting.

Table 2 shows three main patterns. First, *Core WLTS*, with neither PD nor ES, already outperforms Beam Search and ABMCTS-A on average while using substantially fewer tokens, indicating that the main accuracy–efficiency gain comes from the core width-limited backtracking structure rather than optional deployment components. Its per-layer cap distributes search resources across the full response length, allowing each step toward the final answer to receive effective verifier supervision, while backtracking stops unpromising intermediate trajectories and redirects capacity to alternative candidate nodes. Second, ES substantially reduces token usage on canonicalizable-answer bench-

Table 3: Verifier sensitivity on Llama-3.1-8B. The four benchmark columns and "Gen. Avg" report accuracy (%) under a same-backbone generative verifier implemented by Llama-3.1-8B itself. "Cls. Avg" shows the corresponding Skywork-PRM macro-average from Table 1 for direct contrast.

Method	MATH500	Olympiad Bench	AMC23	MBPP	Gen. Avg	Cls. Avg
Best-of-N	49.80	19.41	25.75	66.00	40.24	54.25
Beam Search	50.00	18.96	**33.25**	67.60	42.45	56.93
MCTS	51.40	18.82	31.25	66.80	42.07	56.14
Q*	51.80	20.15	23.75	66.40	40.53	57.66
ABMCTS-A	50.40	18.37	30.00	66.70	41.37	55.00
WLTS	**57.60**	**22.07**	32.50	**68.30**	**45.12**	**62.40**

marks and can also improve accuracy, since recurring answers across completed trajectories are more likely to be correct. This matches its role as an efficiency mechanism for canonicalizable outputs. Third, PD injects exploration diversity by deleting the prefix family of a completed trajectory, freeing width for alternative high-level prefixes and substantially improving accuracy at higher token cost. This supports our expectation that prefix deletion raises the attainable performance ceiling by widening the explored prefix space. Combining ES and PD achieves the best average accuracy while reducing tokens relative to Core WLTS + PD, showing that they are complementary: PD broadens the explored prefix space, whereas ES truncates search once the widened exploration converges on a consistent answer. Overall, ES and PD serve as deployment-time knobs for different accuracy–efficiency requirements. We further analyze prefix-deletion depth h_{del} and agreement threshold M in Appendices C and D.

4.4 Inference-time Scaling Law

We assess inference-time compute scaling by varying each method's search budget and recording macro-averaged accuracy and generated-token usage. The experiment uses Qwen3-14B, with results macro-averaged over AIME24, AIME25, ZebraLogic, and LiveCodeBench; per-benchmark results are provided in Appendix E (Table 6). As shown in Figure 3, all search algorithms generally improve with additional tokens, but WLTS scales most strongly. Increasing the WLTS budget expands exploration under a strict per-layer width cap, so additional computation is distributed across reasoning depths and yields more high-quality completed candidates. In contrast, for tree-search methods such as MCTS, Q*, and ABMCTS-A, additional budget can still be diluted by shallow nodes, leaving less effective search effort for deeper states and producing flatter scaling curves. Beam Search initially improves sharply with larger candidate budgets but eventually bottlenecks: in long-CoT reasoning, retained candidates can concentrate around superficially plausible prefix families, and irreversible pruning prevents recovery from such local optima. This failure mode motivates preserving backtracking in WLTS.

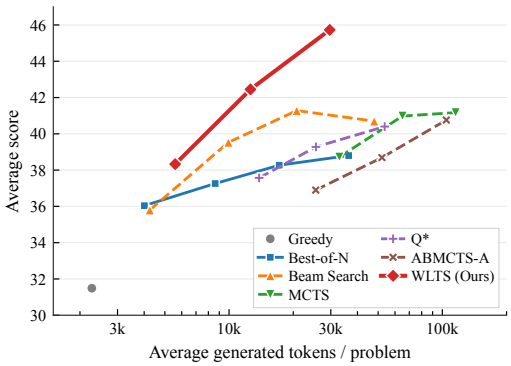


Figure 3: Inference-time scaling on Qwen3-14B.

4.5 Verifier Sensitivity

In the main experiments, each model setting is paired with a strong corresponding verifier: Llama-3.1-8B uses the specialized Skywork-PRM classifier reward model, while the Qwen3 models use their stronger thinking-mode generative verification. Here, we stress-test robustness under a weaker verifier by using Llama-3.1-8B itself as a generative verifier, whose judgments are substantially weaker than Skywork-PRM. Table 3 shows that, as expected, degrading the verifier from the specialized PRM to the self-verifier substantially lowers absolute scores for all methods. Nevertheless, WLTS retains the best macro-average accuracy, ranking first on three of the four tasks and second on the remaining one. This suggests that WLTS is not tied to a particular verifier strength: even with weaker step-level judgments, its width-limited backtracking structure remains robust.

5 Related Work

5.1 Inference-Time Scaling via Sequence Search

Inference-time compute is a key factor for improving LLM reasoning without retraining; scaling inference-time search can rival parameter scaling under fixed compute budgets [20, 12, 28]. The simplest *sequence search* methods sample multiple complete solutions and select an output by reward-model scoring or agreement. Best-of-N returns the reward-model top-ranked solution, with recent work studying optimal test-time compute allocation [20]. Self-consistency selects the majority answer across chain-of-thought samples, improving robustness to stochastic generation [26]. These methods are simple and scale linearly in the number of sampled trajectories, but operate only over full solutions and cannot revise erroneous prefixes once committed.

More structured sequence search scores partial prefixes. Beam search maintains a fixed-size frontier at each step and can incorporate task-specific or learned scoring signals [21]. REBASE can be viewed as a reward-guided extension of beam search, replacing uniform expansion counts per prefix with allocations based on verifier scores [29]. However, pruning in these methods is irreversible: if the frontier collapses onto an incorrect reasoning line early, later evidence cannot recover discarded alternative prefixes. Diversity-promoting decoding methods reduce hypothesis correlation but still remain purely forward decoding methods [11]. WLTS is complementary: it preserves tree-search-style backtracking while using prefix deletion to remove over-explored prefix families after each terminal trajectory, encouraging exploration of distinct high-level plans.

5.2 Tree Search for LLM Reasoning

To enable revision and backtracking, recent methods cast multi-step reasoning as tree search over partial trajectories, such as ToT [32] and GoT [1], often with step-level feedback from process reward models for intermediate correctness [15]. As strong LLMs increasingly serve as generative verifiers, signal quality and granularity have become key determinants of search effectiveness [19, 34]. Compute allocation is central: value-guided pruning/prioritization focus expansions on promising prefixes, while uncertainty-based criteria improve exploration under noisy scores [3, 7]. Planning-oriented methods such as Q* and rStar-Math highlight the gains from exploring and revising intermediate decisions under fixed budgets [24, 4]. Recent work improves the practicality and budget efficiency of LLM tree search [23, 22, 6], including adaptive branching width–depth control [8] and decisions about when to branch [13].

However, adaptive branching control does not by itself enforce a strict per-layer capacity limit and can still allow rapid frontier growth with depth. WLTS shares the goal of budgeted backtracking but differs structurally by enforcing a strict *per-layer* width cap via a dynamic expandability criterion, guaranteeing linear-in-depth growth of generated search-tree nodes. This constraint is orthogonal to the reward model and can be combined with value- or uncertainty-guided selection, pruning, and diversification techniques.

6 Conclusion and Limitations

As LLM responses grow longer, existing tree-search algorithms struggle to distribute limited compute across reasoning depths, with much of the budget consumed by shallow exploration. Motivated by this challenge in long-chain-of-thought reasoning, we presented *Width-Limited Tree Search* (WLTS), an LLM inference search algorithm that reconciles the strengths of sequence search and tree search. WLTS preserves backtracking while provably keeping generated search-tree growth linear in reasoning depth through a strict per-layer width cap enforced by a dynamic expandability criterion. This width-limited structure distributes search budget across the full response length and sustains verifier supervision throughout the final-answer generation process. To accommodate different accuracy–token trade-offs, WLTS further supports two optional components: prefix deletion for exploration diversity and agreement-based early stopping for reducing computation on canonicalizable-output tasks. Extensive experiments consistently demonstrate the effectiveness, generality, and robustness of our method.

Limitations. The current strict width cap remains a relatively static capacity-control mechanism. As LLM responses grow longer, low-quality prefixes may occupy limited per-layer capacity before

sufficient evidence emerges to identify them and reallocate capacity, thereby reducing the effective backtracking space for recovery from early mistakes. A promising direction is to develop more proactive control mechanisms, such as dynamically releasing unpromising nodes during exploration or adaptively adjusting the per-layer width.

Acknowledgments and Disclosure of Funding

Nil.

References

[1] Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of thoughts: Solving elaborate problems with large language models. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 17682–17690, 2024.

[2] Cameron 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.

[3] Julia Grosse, Ruotian Wu, Ahmad Rashid, Cheng Zhang, Philipp Hennig, Pascal Poupart, and Agustinus Kristiadi. Uncertainty-guided likelihood tree search. *arXiv preprint arXiv:2407.03951*, 2024.

[4] Xinyu Guan, Li Lyna Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, Fan Yang, and Mao Yang. rStar-Math: Small LLMs can master math reasoning with self-evolved deep thinking. In *International Conference on Machine Learning (ICML)*, 2025.

[5] Daya Guo, Dejian Yang, Haowei Zhang, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645:633–638, 2025.

[6] Coleman Hooper, Sehoon Kim, Suhong Moon, Kerem Dilmen, Monishwaran Maheswaran, Nicholas Lee, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. ETS: Efficient tree search for inference-time scaling. *arXiv preprint arXiv:2502.13575*, 2025.

[7] Zhiyuan Hu, Chumin Liu, Xidong Feng, Yilun Zhao, See-Kiong Ng, Anh Tuan Luu, Junxian He, Pang Wei Koh, and Bryan Hooi. Uncertainty of thoughts: Uncertainty-aware planning enhances information seeking in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[8] Yuichi Inoue, Kou Misaki, Yuki Imajuku, So Kuroki, Taishi Nakamura, and Takuya Akiba. Wider or deeper? Scaling LLM inference-time compute with adaptive branching tree search. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

[9] Jinhao Jiang, Zhipeng Chen, Yingqian Min, Jie Chen, Xiaoxue Cheng, Jiapeng Wang, Yiru Tang, Haoxiang Sun, Jia Deng, Wayne Xin Zhao, Zheng Liu, Dong Yan, Jian Xie, Zhongyuan Wang, and Ji-Rong Wen. Enhancing LLM reasoning with reward-guided tree search. *arXiv preprint arXiv:2411.11694*, 2024.

[10] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *European Conference on Machine Learning (ECML)*, pages 282–293, Berlin, 2006.

[11] Jiwei Li and Dan Jurafsky. Mutual information and diverse decoding improve neural machine translation. *arXiv preprint arXiv:1601.00372*, 2016.

[12] Xinzhe Li. A survey on LLM test-time compute via search: Tasks, LLM profiling, search algorithms, and relevant frameworks. *Transactions on Machine Learning Research*, 2025.

[13] Xinzhe Li. Chain-in-Tree: Back to sequential reasoning in LLM tree search. In *Findings of the Association for Computational Linguistics (ACL)*, pages 4365–4392, 2026.

[14] Shalev Lifshitz, Sheila A. McIlraith, and Yilun Du. Multi-agent verification: Scaling test-time compute with multiple verifiers. In *Conference on Language Modeling (COLM)*, 2025.

[15] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations (ICLR)*, 2024.

[16] Chris Yuhao Liu, Liang Zeng, Jiacai Liu, Rui Yan, Jujie He, Chaojie Wang, Shuicheng Yan, Yang Liu, and Yahui Zhou. Skywork-Reward: Bag of tricks for reward modeling in LLMs. *arXiv preprint arXiv:2410.18451*, 2024.

[17] OpenAI. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[18] Jing-Cheng Pang, Pengyuan Wang, Kaiyuan Li, Xiong-Hui Chen, Jiacheng Xu, Zongzhang Zhang, and Yang Yu. Language model self-improvement by reinforcement learning contemplation. In *International Conference on Learning Representations (ICLR)*, 2024.

[19] Wenlei Shi and Xing Jin. Heimdall: Test-time scaling on the generative verification. *arXiv preprint arXiv:2504.10337*, 2025.

[20] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. In *International Conference on Learning Representations (ICLR)*, 2025.

[21] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 3104–3112, 2014.

[22] Ante Wang, Linfeng Song, Ye Tian, Baolin Peng, Dian Yu, Haitao Mi, Jinsong Su, and Dong Yu. LiteSearch: Efficient tree search with dynamic exploration budget for math reasoning. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 25318–25326, 2025.

[23] 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 *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 23946–23959, 2025.

[24] Chaojie Wang, Yanchen Deng, Zhiyi Lyu, Liang Zeng, Jujie He, Shuicheng Yan, and Bo An. Q*: Improving multi-step reasoning for LLMs with deliberative planning. *arXiv preprint arXiv:2406.14283*, 2024.

[25] Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-Shepherd: Verify and reinforce LLMs step-by-step without human annotations. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 9426–9439, 2024.

[26] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[27] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[28] Jiaqi Wei, Xiang Zhang, Yuejin Yang, Wenxuan Huang, Juntai Cao, Sheng Xu, Xiang Zhuang, Zhangyang Gao, Muhammad Abdul-Mageed, Laks V. S. Lakshmanan, Chenyu You, Wanli Ouyang, and Siqi Sun. Unifying tree search algorithm and reward design for LLM reasoning: A survey. *arXiv preprint arXiv:2510.09988*, 2025.

[29] Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for LLM problem-solving. In *International Conference on Learning Representations (ICLR)*, 2025.

[30] Jiacheng Xu, Feng Chen, Xiuneng Xu, and Bo An. Entropy-regularized rank-masked policy optimization for test-time reinforcement learning in code generation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 47843–47857, Budapest, Hungary, 2026.

[31] Jiacheng Xu, Wentao Zhang, Zhiyi Lyu, Fuxiang Zhang, Chaojie Wang, Yang Liu, and Bo An. Two-stage reinforcement learning for sound and adversarial test generation in code LLMs. In *Findings of the Association for Computational Linguistics (EMNLP)*, pages 37031–37051, Budapest, Hungary, 2026.

[32] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[33] Fuxiang Zhang, Jiacheng Xu, Chaojie Wang, Ce Cui, Yang Liu, and Bo An. Incentivizing LLMs to self-verify their answers. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

[34] Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. In *International Conference on Learning Representations (ICLR)*, 2025.

A Detailed Complexity Accounting

This appendix gives a more explicit accounting of the generated search tree discussed in Section 3.5. We count search-tree nodes generated for a single input instance, from initialization until WLTS returns an output, including nodes later removed by prefix deletion. This accounting concerns the size of the generated search tree; empirical accuracy–cost tradeoffs are evaluated in Section 4.

For unconstrained tree search, the search space to depth d contains $O(b^d)$ nodes under branching factor b, so exhaustive exploration requires $O(b^d)$ generated nodes. WLTS avoids this exponential depth dependence through the per-layer width cap.

Lemma A.1 (Width invariant)**.** *Throughout WLTS, each layer contains at most W kept search nodes:*

$$|\mathcal{L}_t| \le W, \quad \forall t.$$

Proof. The claim holds at initialization. Consider one expansion that selects a node at depth $t \ge t_{\text{full}}$. The generated child is associated with depth $t+1$; if it remains expandable, it is inserted into layer $\mathcal{L}_{t+1}$. By definition of t_{full}, every layer strictly deeper than t_{full} is not saturated. Since $t+1 \ge t_{\text{full}}+1$, any such insertion preserves $|\mathcal{L}_{t+1}| \le W$. All other layers are unchanged, and prefix deletion only removes kept nodes. Therefore, the width cap is preserved after each expansion. □

Each loop iteration of WLTS generates exactly one step-level node, which is either inserted into a layer $\mathcal{L}_{t+1}$ (*kept*) or added to the terminal set $\mathcal{T}$ (*terminal*). The two corollaries below bound these contributions separately.

Corollary A.2 (Linear-in-depth search-space growth of Core WLTS)**.** *Consider Core WLTS without prefix deletion. Let $D_{\max}$ be the maximum reasoning depth, W the per-layer width cap, and N the terminal-candidate target. Then the total number of search-tree nodes generated for a single input instance is bounded by*

$$G_{\text{core}} \le WD_{\max} + N.$$

Thus, for fixed W and N, Core WLTS has linear search-space growth in reasoning depth.

Proof. Each generated node is either kept or terminal. Without deletion, no layer capacity is reopened after a kept node is generated at that depth. By Lemma A.1, each layer can therefore receive at most W kept generated nodes, so across depths $1, \dots, D_{\max}$ the total number of kept generated nodes is at most $WD_{\max}$. The number of terminal nodes is bounded by $|\mathcal{T}| \le N$ at termination, since the loop breaks once $|\mathcal{T}| \ge N$. Summing the two contributions gives $G_{\text{core}} \le WD_{\max} + N$. □

Corollary A.3 (Linear-in-depth search-space growth with deletion)**.** *Consider WLTS with prefix deletion. Let C be the number of deletion operations during this invocation. Then the total number of generated search-tree nodes is bounded by*

$$G_{\text{del}} \le (C+1)WD_{\max} + N.$$

Since prefix deletion is triggered only after a terminal trajectory is found and WLTS collects at most N terminal trajectories, $C \le N$. Therefore,

$$G_{\text{del}} \le (N+1)WD_{\max} + N = O(NWD_{\max}).$$

Thus WLTS with prefix deletion also has linear search-space growth in reasoning depth under fixed search-budget parameters.

Proof. Each generated node is either kept or terminal; we bound the two contributions separately. For kept nodes, fix any depth t. Before the first deletion operation, the width invariant ensures that at most W kept nodes can be generated at depth t. Each deletion operation can free capacity in that layer, after which at most another W kept nodes can be generated at the same depth before the next deletion. Thus, after C deletion operations, the total number of kept generated nodes at depth t is at most $(C+1)W$. Summing over all depths gives at most $(C+1)WD_{\max}$ kept nodes. For terminal nodes, $|\mathcal{T}| \le N$ at termination, since the loop breaks once $|\mathcal{T}| \ge N$. Summing the two contributions gives $G_{\text{del}} \le (C+1)WD_{\max} + N$. Finally, prefix deletion is triggered only after a terminal trajectory is collected, and the algorithm collects at most N terminal trajectories; hence $C \le N$. Substituting this bound gives the stated $O(NWD_{\max})$ result. □

B Implementation Details

B.1 Hyperparameters

Unless otherwise specified, we use the default sampling parameters provided by each model.

For Qwen3-8B and Qwen3-14B, we set the temperature to 0.6 for the single-pass sampling baseline (no search) and to 0.7 for search algorithms to increase diversity.

For Llama-3.1-8B-Instruct, we use a temperature of 0 for the single-pass baseline (no search) and 0.6 for search algorithms to promote diversity.

The maximum answer length is set to 4096 tokens for Llama-3.1-8B-Instruct and 8192 tokens for Qwen3-8B and Qwen3-14B. For Llama-3.1-8B-Instruct, steps are segmented by line breaks. For Qwen3-8B and Qwen3-14B, we use a token-based segmentation scheme: each step contains 128 tokens for code-generation tasks and 512 tokens for other tasks, reflecting the difference in average response length across task types.

For Qwen3-8B and Qwen3-14B, we use non-thinking mode in the generator and thinking mode in the verifier, as an accurate verifier is required for verifier-guided search algorithms.

For the main comparison, tree-search baselines (MCTS, Q*, and ABMCTS-A) use a fixed selection budget $S = 128$ and target up to $N = 16$ terminal candidate trajectories.

All step-level methods share the proposer, verifier, step segmentation, and sampling parameters. They receive no separately trained task-specific value model or external feedback unavailable to the other methods.

B.2 Benchmark Details

For datasets with fewer than 100 problems, including AMC23, AIME24, and AIME25, we run each benchmark 10 times and report the average result. For all other datasets, we run multiple times to ensure the total sample size is at least 500, and then take the average. LiveCodeBench consists of code submissions collected from October 2024 to February 2025. For ZebraLogic, we use the grid mode setting.

B.3 Computation Cost

This paper uses 80GB A100 GPUs for inference, with a cumulative inference time of approximately 7,500 hours across all experiments.

We use a multithreaded parallel approach to send multiple queries simultaneously. As a result, the total inference time on each dataset is approximately proportional to the total number of tokens processed. Specific wall-clock case experiments are shown in Table 8.

C Diversity Enhancement: Detailed Analysis

Protocol. Prefix deletion is intended to prevent WLTS from over-investing in a single high-scoring prefix family (Section 3.3). To isolate this mechanism, we sweep the deletion depth $h_{\mathrm{del}} \in \{1, 2, 4, 8\}$ while fixing $(W, N, M) = (8, 16, 2)$, and evaluate Qwen3-8B and Qwen3-14B on AIME24, AIME25, and LiveCodeBench. We report both per-benchmark results and macro-averaged accuracy/token usage, since changing the deletion depth affects not only answer quality but also how quickly WLTS reaches repeated terminal answers.

Results. Figure 4 and Table 4 reveal two consistent patterns across both models. First, token usage decreases monotonically as h_{del} grows: deeper deletion truncates only narrow subtrees and leaves most of the explored prefix structure intact, so the freed capacity is spent extending nearby continuations rather than opening new high-level branches.

Second, accuracy is non-monotonic in h_{del} and peaks at an intermediate value, confirming that prefix deletion functions as a diversity control rather than a pure pruning rule. Very shallow deletion erases broad prefix families together with their useful nearby continuations and forces exploration into distant regions that may not warrant the cost; very deep deletion preserves those continuations but exerts too little pressure to redirect search toward genuinely different high-level plans, so the algorithm keeps refining variants of an already-explored prefix. The intermediate setting balances these effects, retaining enough nearby continuations to capitalize on already-vetted prefix structure while still freeing capacity for substantially different alternatives—precisely the regime where prefix deletion is intended to operate. For deployment, choose W according to the inference budget and start with $h_{\text{del}} = 1$–2, increasing it when token cost is the limiting factor.

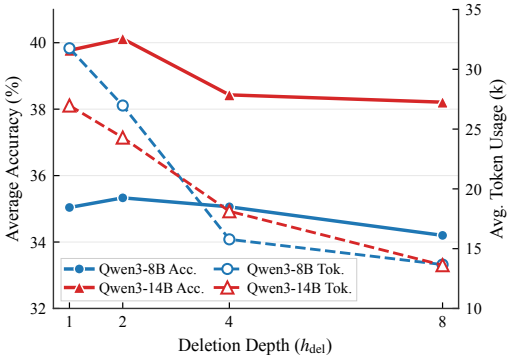


Figure 4: Effect of prefix deletion depth h_{del} on macro-averaged accuracy (solid lines, left axis) and token usage (dashed lines, right axis), evaluated on Qwen3-8B and Qwen3-14B over AIME24, AIME25, and LiveCodeBench.

Table 4: Prefix deletion depth ablation (Qwen3-8B and Qwen3-14B). Acc/Tok reports accuracy (%) and average token usage (k).

Model	h_{del}	AIME24	AIME25	LiveCodeBench	Average
Qwen3-8B	1	42.67/42.91	29.33/38.21	**33.13**/14.16	35.04/31.76
	2	**44.00**/34.35	30.67/31.42	31.33/15.15	**35.33**/26.97
	4	41.33/18.48	**31.33**/17.19	32.53/11.67	35.06/15.78
	8	40.00/15.23	30.67/15.72	31.93/10.05	34.20/13.67
Qwen3-14B	1	**46.00**/32.58	35.33/33.70	37.99/14.64	39.77/26.97
	2	43.33/32.78	**36.67**/29.26	**40.36**/10.83	**40.12**/24.29
	4	40.67/22.02	**36.67**/24.02	37.95/8.35	38.43/18.13
	8	44.67/16.08	32.00/16.50	37.95/8.27	38.21/13.62

D Early Stopping: Detailed Analysis

Protocol. We ablate the agreement threshold M in the early-stopping rule (Section 3.4) on AIME24 and AIME25, where final answers can be canonicalized. For each setting, we report final accuracy, generated-token usage, the probability that early stopping is triggered, and the accuracy conditioned on early stopping. These metrics separate two effects: how often the search terminates before collecting all candidates, and how reliable the repeated answer is when termination occurs.

Table 5: Early stopping ablation on math benchmarks. Acc/Tok reports accuracy (%) and average token usage (k). ES/ES Acc reports the early-stop rate (%) and the accuracy conditioned on early stop (%). Average is the arithmetic mean over AIME24 and AIME25.

Model	M	AIME24		AIME25		Average	
		Acc/Tok	ES/ES Acc	Acc/Tok	ES/ES Acc	Acc/Tok	ES/ES Acc
Qwen3-8B	2	42.67/42.91	97.33/43.84	29.33/38.21	96.00/29.86	36.00/40.56	96.67/36.85
	3	44.00/68.26	84.67/50.39	29.33/54.13	80.67/35.54	36.67/61.20	82.67/42.97
	4	**44.33**/78.20	75.33/**50.44**	**30.00**/66.64	67.33/**41.58**	**37.17**/72.42	71.33/**46.01**
Qwen3-14B	2	**46.00**/32.58	96.00/46.53	35.33/33.70	97.33/34.25	40.67/33.14	96.67/40.39
	3	44.00/60.95	88.67/49.62	37.33/58.50	84.00/**42.06**	40.67/59.73	86.34/45.84
	4	45.33/80.84	75.33/**54.87**	**37.67**/66.74	78.00/41.88	**41.50**/73.79	76.67/**48.38**

Results. Table 5 exhibits two coupled effects of the agreement threshold M. First, raising M shifts the early-stop rate downward and the conditional early-stop accuracy upward, confirming that repeated terminal answers become increasingly reliable as more independent trajectories agree on them, since a coincidental majority among incorrect answers is unlikely under sufficient agreement. Second, this added reliability is purchased at a disproportionate token cost: each additional required agreement compels WLTS to complete more candidates before consensus can form, and most of those extra trajectories are produced after the would-be majority answer has already emerged, so the search spends additional compute on confirmation rather than on discovering new answers. Consequently, the marginal final-accuracy gain from larger M is small relative to the token overhead, and M functions primarily as a deployment-time knob: small values are appropriate when budget pressure dominates, while larger values are justified only when a modest accuracy gain warrants substantially more inference compute.

E Inference-Time Scaling: Detailed Results for Qwen3-14B

Protocol. Table 6 reports the detailed per-benchmark results underlying the inference-time scaling analysis in Section 4.4 (Figure 3). We sweep compute for sequence search (BON and Beam Search), tree search (MCTS, Q*, and ABMCTS-A), and WLTS on Qwen3-14B. Acc/Tok reports accuracy (%) and average generator token usage (k, in thousands); all runs use temperature t=0.7 and maximum generation length l=8192.

Table 6: Inference-time scaling ablation on Qwen3-14B: detailed results for sequence search (BON/beam), tree search (MCTS/Q*/ABMCTS-A), and WLTS under different compute settings. The value before the slash is the accuracy, and the value after the slash is the token usage (k). All four benchmark columns are completed for the listed configurations.

Method	Compute setting	AIME24	AIME25	LiveCodeBench	ZebraLogic	Average
BON	N=2	35.33/5.58	30.00/5.21	30.72/0.97	48.10/4.28	36.04/4.01
	N=4	37.33/12.63	28.67/11.22	33.13/2.03	49.90/8.57	37.26/8.61
	N=8	42.00/24.98	30.67/22.02	30.72/4.65	49.70/17.09	38.27/17.19
	N=16	40.67/56.16	32.00/46.06	31.93/9.09	50.60/34.22	38.80/36.38
Beam	W=2, b=1	34.00/6.06	28.67/5.50	30.72/1.18	49.80/4.27	35.80/4.25
	W=2, b=2	38.00/14.47	<u>34.00</u>/12.93	33.73/3.44	52.40/8.86	39.53/9.93
	W=4, b=2	42.67/30.60	32.00/27.92	34.94/6.13	55.50/18.38	41.28/20.76
	W=4, b=4	40.00/76.74	33.33/62.52	34.34/18.55	55.10/33.84	40.70/47.91
MCTS	S=32, N=4	40.67/40.46	26.67/41.99	33.73/8.29	53.80/40.81	38.72/32.89
	S=64, N=8	41.33/86.21	32.67/82.58	33.73/22.65	<u>56.20</u>/68.71	40.98/65.04
	S=128, N=16	42.00/155.00	33.33/152.75	34.94/40.98	54.40/112.89	41.17/115.41
Q*	S=32, N=4	36.66/14.72	30.67/13.77	33.13/9.38	49.80/17.50	37.57/13.84
	S=64, N=8	38.33/31.54	31.33/28.45	36.75/21.25	50.70/20.98	39.28/25.55
	S=128, N=16	38.00/57.28	33.66/51.57	**38.55**/43.42	51.40/62.38	40.40/53.66
ABMCTS-A	S=64, N=4	37.33/27.81	29.33/28.71	31.33/19.66	49.60/25.78	36.90/25.49
	S=128, N=8	38.00/58.30	29.33/57.79	34.94/41.22	52.50/51.09	38.69/52.10
	S=256, N=16	42.67/117.78	32.00/114.20	37.35/83.27	51.00/102.26	40.76/104.38
WLTS	W=2, N=2	38.67/8.04	31.33/7.47	30.12/1.11	53.20/5.73	38.33/5.59
	W=4, N=4	<u>43.33</u>/17.42	<u>34.00</u>/16.62	37.35/2.10	55.10/14.27	<u>42.45</u>/12.60
	W=8, N=16	**46.00**/32.58	**35.33**/33.70	<u>37.99</u>/14.64	**63.60**/37.69	**45.73**/29.65

Results. Table 6 reveals distinct scaling behaviors across method families, all of which trace back to how each algorithm uses additional budget. Best-of-N samples whole trajectories, so additional budget broadens the candidate pool but cannot revise erroneous prefixes shared across samples; its curve flattens once the dominant failure modes saturate the candidate set. Beam Search benefits from step-level scoring and improves sharply at moderate widths, but at larger widths its irreversible pruning becomes the binding constraint: extra capacity is consumed by minor variants of the surviving

frontier and cannot recover from prematurely committed prefixes, so accuracy stalls or even regresses despite the larger budget. Standard tree search restores backtracking but spends most of its added budget on shallow nodes, since unconstrained branching leaves deeper layers visit-starved; MCTS therefore consumes substantially more tokens for marginal gains, and Q* and ABMCTS-A trim some of this cost but inherit the same shallow bias and remain below WLTS at comparable budgets. WLTS exhibits the steepest and most sustained improvement because the per-layer width cap forces each additional expansion to deepen an existing path or open a new prefix family rather than thicken an already-saturated layer, so extra compute translates directly into deeper search and more high-quality completed candidates while backtracking preserves the recovery options that sequence search lacks. The advantage is broad rather than benchmark-specific: WLTS leads on the majority of benchmarks at every budget level, with the only crossover occurring on LiveCodeBench at high budget, where Q* attains a marginally higher score using several times more tokens.

F Token-Budget-Matched Comparison

Protocol. The main comparison in Table 1 matches the target number of terminal candidates, but realized token usage differs because each search method allocates budget differently. WLTS may stop early on canonicalizable tasks, whereas tree-search baselines can spend several times more tokens before producing the same number of terminal candidates. To complement the candidate-count-matched protocol, we additionally compare methods within a similar token range on Qwen3-14B.

We add a Best-of-N run with $N{=}10$ so that its average token usage falls near the Core/Full WLTS range. For the remaining baselines, we select the configurations from Table 6 with average token usage closest to this range: Beam Search with $W{=}4, b{=}2$, MCTS with $S{=}32, N{=}4$, Q* with $S{=}64, N{=}8$, and ABMCTS-A with $S{=}64, N{=}4$, where S denotes the number of tree-search selection iterations. We also include Core WLTS, which disables both prefix deletion and agreement-based early stopping, to isolate the effect of the width-limited backtracking structure.

Table 7: Token-budget-matched comparison on Qwen3-14B. Each cell reports accuracy (%) / generated tokens (k) per problem. BON uses an additional $N{=}10$ run, and the other baselines are selected from Table 6 to have token usage near the Core/Full WLTS range. "Core WLTS" disables both prefix deletion and agreement-based early stopping; all listed configurations are complete.

Method	AIME24	AIME25	LiveCodeBench	ZebraLogic	Average
Best-of-N $N{=}10$	40.67 / 31.96	31.33 / 27.93	35.54 / 7.14	50.00 / 21.49	39.39 / 22.13
Beam Search $W{=}4, b{=}2$	<u>42.67</u> / 30.60	32.00 / 27.92	34.94 / 6.13	55.50 / 18.38	41.28 / 20.76
MCTS $S{=}32, N{=}4$	40.67 / 40.46	26.67 / 41.99	33.73 / 8.29	53.80 / 40.81	38.72 / 32.89
Q* $S{=}64, N{=}8$	38.33 / 31.54	31.33 / 28.45	36.75 / 21.25	50.70 / 20.98	39.28 / 25.55
ABMCTS-A $S{=}64, N{=}4$	37.33 / 27.81	29.33 / 28.71	31.33 / 19.66	49.60 / 25.78	36.90 / 25.49
Core WLTS	41.33 / 27.28	<u>32.33</u> / 25.88	<u>37.35</u> / 4.09	<u>55.80</u> / 26.51	<u>41.70</u> / 20.94
WLTS	**46.00** / 32.58	**35.33** / 33.70	**37.99** / 14.64	**63.60** / 37.69	**45.73** / 29.65

Results. Table 7 confirms that the WLTS advantage is not an artifact of the candidate-count-matched protocol. Core WLTS already matches or exceeds the strongest non-WLTS baseline at essentially the same average token budget, isolating the per-layer width cap and backtracking structure as the primary source of the accuracy–efficiency gain rather than the optional deployment components. The underlying mechanism is the same one identified in the main text: a strict per-layer cap forces extra expansions to deepen the search rather than thicken shallow layers, so under any fixed token budget a larger fraction of compute lands on deeper steps where verifier supervision is otherwise weakest, while backtracking allows that deeper compute to revise rather than only extend earlier prefixes. Adding prefix deletion and agreement-based early stopping shifts Full WLTS into a slightly higher token regime but yields a substantially larger accuracy gain, indicating that the optional components broaden the prefix space and trim redundant exploration without altering the core advantage. The fact that WLTS leads on every benchmark in this table further suggests that the gain is structural rather than benchmark-specific, mirroring the picture from the candidate-count-matched protocol.

G Runtime Metric

The main results in Table 1 report computational cost via generator token usage, which gives a hardware-agnostic view but is not equivalent to wall-clock latency. Latency also depends on the serving stack and the parallelism structure of each search method: in modern LLM serving, search-based methods can still benefit from concurrent requests and prefix KV-cache reuse, but their parallelism is generally less direct than Best-of-N due to stronger inter-step dependencies. Best-of-N is embarrassingly parallel across the N trajectories; sequence search such as Beam Search exposes width-level parallelism per step but is sequential across steps; and backtracking-based tree search (MCTS, Q*, WLTS) introduces additional inter-step dependencies that limit how much computation can be overlapped in time.

Setup. To complement the token-usage view in Table 1, we additionally measure wall-clock runtime on Llama-3.1-8B (Skywork-PRM verifier) under the same multithreaded $4\times$A800 setup used for the main experiments, with all other settings unchanged. Table 8 reports per-benchmark accuracy, generator tokens, and wall-clock time per problem on MATH500, OlympiadBench, AMC23, and MBPP, together with their macro-averages.

Table 8: Per-benchmark wall-clock runtime on Llama-3.1-8B (Classifier RM setting) under a multi-threaded $4\times$A800 setup. Acc. is accuracy (%); Tok. is the average generated tokens per problem (k); Time is the average wall-clock completion time per problem (seconds). The Average column reports macro-averaged values across the four benchmarks; parentheses next to accuracy indicate absolute improvements over Best-of-N (in percentage points), and parentheses next to Tok. and Time indicate cost relative to Best-of-N (= 100%).

Method	Metric	MATH500	Olympiad Bench	AMC23	MBPP	Average
Best-of-N	Acc.(%)	66.20	28.30	50.00	72.50	54.25
	Tok.(k)	12.68	23.16	20.24	4.42	15.13 (100%)
	Time(s)	1.06	1.76	1.62	0.32	1.19 (100%)
Beam Search	Acc.(%)	70.20	33.33	52.50	71.70	56.93 (+2.68)
	Tok.(k)	9.87	14.47	13.08	4.10	10.38 (69%)
	Time(s)	2.76	3.84	3.31	0.73	2.66 (224%)
MCTS	Acc.(%)	66.40	30.67	55.00	72.50	56.14 (+1.89)
	Tok.(k)	68.93	86.37	91.65	13.39	65.09 (430%)
	Time(s)	16.21	20.86	21.77	3.46	15.58 (1309%)
Q*	Acc.(%)	69.80	32.74	55.00	73.10	57.66 (+3.41)
	Tok.(k)	57.15	60.39	62.46	14.22	48.56 (321%)
	Time(s)	14.23	14.43	15.34	12.40	14.10 (1185%)
ABMCTS-A	Acc.(%)	66.20	30.39	52.50	70.90	55.00 (+0.75)
	Tok.(k)	38.16	42.48	41.34	18.39	35.09 (232%)
	Time(s)	15.71	17.39	17.23	7.47	14.45 (1214%)
WLTS	Acc.(%)	**72.40**	**36.30**	**65.00**	**75.90**	**62.40** (+8.15)
	Tok.(k)	16.98	30.54	30.05	7.57	21.29 (141%)
	Time(s)	**6.81**	**11.76**	**11.43**	3.17	**8.29** (697%)

Results. Table 8 matches the qualitative picture from the token-cost view but exposes two effects that token counts alone cannot reveal. Among backtracking-based methods, WLTS achieves the best accuracy–latency trade-off because the per-layer width cap upper-bounds the work performed at each depth and therefore the number of sequential expansion rounds needed to fill the search tree, whereas unconstrained tree search must service many more shallow expansions before producing comparable terminal candidates. Best-of-N remains the fastest in raw latency due to its embarrassingly parallel structure across complete trajectories, and Beam Search retains a latency edge through per-step batching of frontier candidates; both, however, omit backtracking entirely and pay for that absence in final accuracy. The takeaway is therefore not that WLTS is universally faster than non-backtracking baselines, but that the per-layer width cap turns backtracking from a latency liability into a controllable cost, making WLTS the most practical option whenever recovery from erroneous prefixes is required.

The three unconstrained tree-search baselines (MCTS, Q*, and ABMCTS-A) all fall in a similar 14–16 s wall-clock band per problem despite their different expansion heuristics, indicating that their shared inter-step dependency structure—rather than per-method micro-optimizations—is what dominates runtime; WLTS sits below this band at 8.29 s while still delivering the highest accuracy.

Each generated step-level node triggers a proposer call and a verifier call, with decoding bounded by the proposer's per-step token budget and the generative verifier's response cap. Longer prefixes increase prefill computation, while prefix caching allows shared prefixes to reuse their keys and values. Profiling WLTS under its main configuration on Qwen3-8B and AIME25 shows that total prefill tokens are 3.67 times total generated tokens, yet the prefill-to-decode time ratio is only 0.0065. Decoding therefore dominates the measured runtime, supporting generated-node count as a practical compute proxy in this setting. This correspondence weakens when decoding no longer dominates, such as with very short generations and disproportionately long prompts, or when prefix-cache reuse differs sharply across methods.

H Comparison with REBASE and ABMCTS-M

REBASE [29] is a reward-guided extension of beam search that allocates the remaining sampling budget among prefixes at each depth according to softmax-normalized rewards. We evaluate it on both Qwen3 models with a layer-level expansion budget of 16. Table 9 compares it with Beam Search and WLTS. The results show that WLTS obtains higher mean accuracy with fewer generator tokens than REBASE on both models.

Table 9: Comparison with REBASE. Cells report accuracy (%) / generator tokens (k).

Model	Method	AIME24	AIME25	ZebraLogic	LiveCodeBench	Mean
Qwen3-8B	Beam Search	32.67/72.8	27.33/55.2	45.80/52.6	31.93/16.5	34.43/49.26
	REBASE	35.33/57.0	28.67/50.6	44.00/46.8	30.12/13.7	34.53/42.03
	WLTS	42.67/42.9	29.33/38.2	48.40/36.4	33.13/14.2	38.38/32.92
Qwen3-14B	Beam Search	40.00/76.7	33.33/62.5	55.10/33.8	34.34/18.6	40.70/47.91
	REBASE	42.67/55.5	32.67/50.0	52.60/36.7	34.94/13.3	40.72/38.88
	WLTS	46.00/32.6	35.33/33.7	63.60/37.7	37.99/14.6	45.73/29.65

We additionally evaluate another ABMCTS variant, ABMCTS-M [8], with Qwen3-8B under the shared-verifier step-level interface. The two variants trade wins across tasks and have similar macro-averages, while WLTS has the highest macro-average (Table 10).

Table 10: Accuracy (%) of ABMCTS variants and WLTS on Qwen3-8B.

Method	AIME24	AIME25	LiveCodeBench	ZebraLogic	Mean
ABMCTS-M	34.00	26.67	27.71	44.80	33.30
ABMCTS-A	32.67	29.33	29.52	40.80	33.08
WLTS	42.67	29.33	33.13	48.40	38.38

I Additional Budget Tiers

Table 11 reports four operating tiers on AIME24 with Qwen3-14B. The tiers group runs with broadly comparable realized generator-token costs. WLTS has the highest observed accuracy in each tier.

J Maximum Verifier-Supervised Depth

We record the deepest verifier-supervised node on each of 1,000 ZebraLogic problems under the main-table Qwen3-14B setting. Table 12 reports the mean maximum depth for small, medium, and large grids. The methods reach similar depths on small grids, while WLTS reaches deeper supervised states on medium and large grids.

Table 11: Multi-budget comparison. Cells report accuracy (%) / generator tokens (k).

Method	Small	Medium	Large	Extra Large
Best-of-N	36.33/9.01	37.66/19.37	37.33/38.43	42.67/75.96
Beam Search	38.00/14.47	38.67/22.97	42.67/30.60	40.00/76.74
MCTS	36.67/10.88	37.66/21.03	40.67/40.46	41.33/86.21
Q*	35.33/10.52	34.33/18.83	38.33/31.54	39.33/71.29
WLTS	38.67/8.04	43.33/17.42	46.00/32.58	48.67/67.53

Table 12: Mean maximum verifier-supervised depth (steps) on ZebraLogic, grouped by grid size.

Method	Small	Medium	Large
ABMCTS-A	2.1	2.6	2.8
MCTS	2.3	3.9	4.7
Q*	2.5	4.9	7.6
WLTS	2.4	5.7	10.4

K Step Segmentation

We vary only step length on Qwen3-14B for AIME24 and AIME25 (Table 13). Finer segmentation improves the reported mean accuracy while increasing generator-token usage. The 512-token setting provides a lower-cost operating point among these configurations.

Table 13: Step-length sensitivity on Qwen3-14B.

Step length (tokens)	AIME24 (%)	AIME25 (%)	Mean Acc. (%)	Mean Tok. (k)
128	48.67	38.00	43.34	72.4
256	47.33	36.00	41.67	48.0
512	46.00	35.33	40.67	33.1

L Selection Priority

All main WLTS experiments use $c_{\mathrm{uct}} = 0$, giving value-only selection with ties broken toward deeper nodes. Table 14 compares priorities on Qwen3-14B. Overall, all three settings perform well.

Table 14: Selection-rule sensitivity on Qwen3-14B.

Selection rule	AIME24 (%)	AIME25 (%)	Mean Tok. (k)
Value only, $c_{\mathrm{uct}} = 0$	46.00	35.33	33.1
Global bonus, $c_{\mathrm{uct}} = 0.5$	44.67	34.00	35.7
Global bonus, $c_{\mathrm{uct}} = 1.0$	45.00	35.67	31.9

M Generative Verifier Prompts

Prompt Template for Generative Verifier

```
[User]
{QUESTION}

[Assistant]
{CANDIDATE_REASONING_OR_SOLUTION}

[User]
You are a careful and precise verifier. Your task is to assess whether the
assistant's reasoning so far is logically and mathematically correct. You
must not attempt to solve the problem yourself or continue the reasoning.
Only check whether the existing reasoning is correct step by step. At the end
of your response, on a new line, write exactly:
"Is the existing reasoning correct (Yes/No)? X"
where X is either Yes or No. Only write Yes if all steps so far are logically
correct.
```

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes].

   Justification: The abstract and introduction state the algorithmic contribution, the linear-in-depth search-space claim, and the empirical scope over mathematical, coding, and logical reasoning benchmarks. These claims are supported by the method and complexity analysis in Sections 3.5 and A, and by the experiments in Section 4.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes].

   Justification: The paper includes a "Conclusion and Limitations" section that discusses the static nature of the current strict width cap and how low-quality prefixes can occupy limited per-layer capacity before being identified. It also discusses accuracy–efficiency trade-offs and runtime considerations in Appendices D, F, and G.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes].

   Justification: The main theoretical claim concerns linear-in-depth generated search-tree growth under the WLTS width cap. Section 3.5 states the result and Appendix A provides the detailed assumptions, lemma, corollaries, and proofs.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes].

   Justification: The paper specifies the evaluated models, verifiers, benchmarks, search baselines, candidate-count protocol, token accounting, and main hyperparameters in Sections 4 and Appendix E. Additional implementation details, benchmark repetition rules, compute information, and verifier prompts are provided in the appendix. We also submit the code as supplementary material.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes].

   Justification: All datasets used in the paper are publicly available benchmarks, and the model and benchmark choices are identified in Section 4 and the appendix. We also provide the implementation code in the appendix/supplementary material, including the components needed to reproduce the main experimental results.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes].

   Justification: The paper is an inference-time search study and does not train new models. It specifies model/verifier choices, benchmarks, search budgets, sampling temperatures, answer-length limits, step segmentation, repetition rules, and verifier prompts in Section 4 and Appendix sections on implementation details and generative verifier prompts.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No].

Justification: Running the full evaluation repeatedly is computationally expensive: the reported experiments require approximately 7,500 hours of inference on 80GB A100 GPUs. This work is an inference-time scaling study, where each problem-level result already aggregates up to $N{=}16$ searched candidate trajectories; for datasets with fewer than 100 problems, we repeat each benchmark 10 times, and for the remaining datasets we repeat runs until the total number of evaluated samples is at least 500. Similar inference-time search/scaling papers typically do not report this statistical metric.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes].

   Justification: Appendix computation-cost details report the GPU class and memory, cumulative inference time, and multithreaded execution strategy, while tables throughout the experiments report generated-token usage. Appendix G further reports wall-clock runtime under a multithreaded $4\times$A800 setup for representative methods.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes].

   Justification: The work evaluates inference-time search algorithms on existing reasoning benchmarks and does not involve human subjects, private data, or deployment decisions. No deviations from the NeurIPS Code of Ethics are identified in the manuscript.

Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [No].

    Justification: The paper discusses technical limitations and compute trade-offs, but it does not include a dedicated broader-impact discussion covering both positive and negative societal impacts. The work is foundational inference-time search research, and any societal effects would arise through downstream uses of stronger LLM reasoning systems.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A].

    Justification: The paper does not introduce or release a new pretrained language model, image generator, scraped dataset, or other high-risk asset. It proposes an inference-time search algorithm evaluated using existing models and benchmarks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.

- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes].

    Justification: All assets used in this paper—including the base models (Llama-3.1-8B-Instruct, Qwen3-8B, Qwen3-14B), the reward model (Skywork-o1-Open-PRM-Qwen-2.5-7B), and the evaluation benchmarks (MATH500, OlympiadBench, AMC23, MBPP, AIME24, AIME25, ZebraLogic, LiveCodeBench)—are openly available, properly credited via citation, and used in accordance with their respective licenses and intended research use.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A].

    Justification: The paper introduces an algorithmic method rather than a new dataset, benchmark, pretrained model, or other standalone asset. No new asset release is described in the current manuscript.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A].

Justification: The research does not involve crowdsourcing experiments or human-subject studies. It evaluates LLM inference-time search on existing reasoning benchmarks.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A].

    Justification: The paper does not involve crowdsourcing or human-subject research, so IRB approval or equivalent review is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes].

    Justification: We describe the LLMs used as proposers and generative verifiers, including their roles, model configurations, and verifier prompts, in the method and experimental setup sections and the appendix.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.