Chain-in-Tree: Back to Sequential Reasoning in LLM Tree Search

Xinzhe Li
Independent Researcher
sergioli212@outlook.com

Abstract

Test-time scaling improves large language models (LLMs) on long-horizon reasoning tasks by allocating more compute at inference. LLM inference via tree search (LITS) achieves strong performance but is highly inefficient. We propose **Chain-in-Tree (CiT)**, a plug-in framework that decides *when* to branch during search instead of expanding at every step. CiT introduces lightweight **Branching Necessity (BN)** evaluations, including **BN-DP** (direct prompting) and **BN-SC** (self-consistency). Integrated into Tree of Thoughts, ReST-MCTS, and RAP, BN-DP reduces token generation, model calls, and runtime by **75–85%** on GSM8K and Math500, with often negligible or no accuracy loss. BN-SC typically yields substantial savings (up to 80%) generally but shows instability in 1–4 out of 14 settings, caused by a small subset of examples that produce extremely long reasoning steps. We theoretically prove that BN-DP never increases policy invocations and release unified implementations applicable across LITS frameworks. The full codebase is publicly available at https://github.com/xinzhel/chain_in_tree.

1 Introduction

Large language models (LLMs) excel at tasks such as mathematical and commonsense reasoning, but their performance often improves further when additional test-time compute is allocated. Recent work has shown that *test-time scaling* enables LLMs to explore multiple reasoning paths, thereby making better use of knowledge they already possess (Sprague et al., 2025). Among test-time scaling methods, tree-search–based approaches have achieved state-of-the-art results on benchmarks like GSM8K and Math500, by treating reasoning as sequential decision-making and exploring multiple candidate trajectories (Zhang et al., 2024a; Yao et al., 2023a). Such approaches pivot LLMs—otherwise weak in zero-shot planning—toward effective performance on planning problems (Valmeekam et al., 2023).

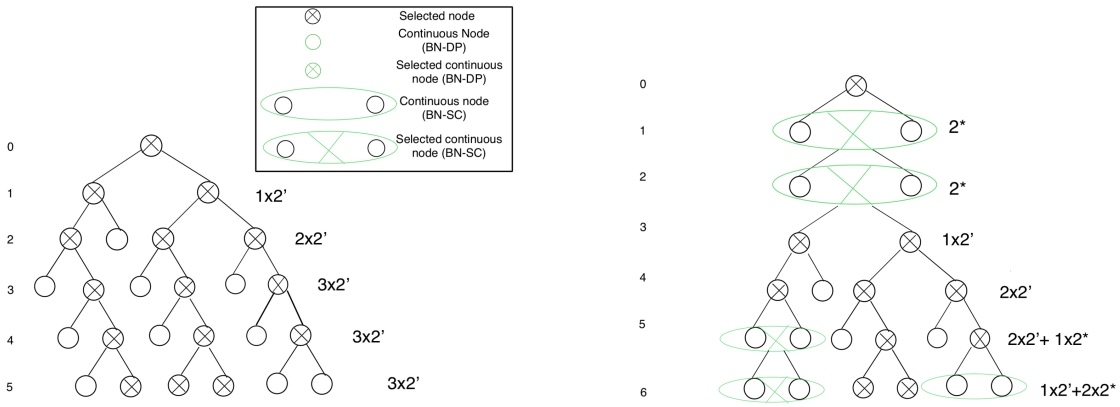
However, tree-search–based frameworks are highly inefficient: compared to simpler iterative prompting methods, they are often 10–20 times slower (Chen et al., 2024). A central limitation is that they operate at a fixed granularity of reasoning steps. In some cases, this granularity is enforced explicitly, for example by treating each subquestion–answer pair as a node (Hao et al., 2023). Other approaches, such as MCTS with free-form thoughts (Zhang et al., 2024a; Yao et al., 2023a), allow more flexibility but still assume that every generated thought must branch independently. In practice, however, a reasoning step in mathematics may correspond to a single operation or a tightly coupled set of operations, while in domains with deterministic action spaces (e.g., board games) the step granularity is even more strictly defined. This rigidity forces branching even on trivial steps, resulting in excessive LLM calls during both expansion and simulation. We address this limitation by proposing **Chain-in-Tree (CiT)**, a plug-in for LLM-in-the-loop tree search that adaptively decides *when* branching is necessary.

The key idea is to introduce continuous nodes chained together: steps deemed confident or routine by the model are chained sequentially, while branching is reserved only for uncertain points where exploration is valuable. Figure 1 provides a high-level illustration of how CiT differs from an existing LITS framework. This reduces unnecessary expansion, lowers inference costs, and preserves the search capacity of existing frameworks.

Contributions. Our contributions are as follows:

- We formulate the decision of when to branch as a new component in tree search and propose two lightweight **Branching Necessity (BN) evaluation methods**: Direct Prompting (BN-DP) and Self-Consistency (BN-SC). These

Findings of the Association for Computational Linguistics: ACL 2026, pages 4365–4392



(a) ToT-Beam Search.

(b) ToT-Beam Search + Chaining (BN-SC).

Figure 1: Tree-of-Thoughts Beam Search (ToT-BS) vs. ToT-BS with Chaining (BN-SC). Primes ′ indicate beam sampling size; asterisks $*$ indicate BN-judge sampling size. See Appendix D for BN-DP.

methods allow LLMs to autonomously determine whether branching is needed at each step. The algorithmic design enabled by introducing this additional phase is detailed in Appendix A.

- We provide a **theoretical guarantee** that BN-DP never increases runtime relative to the baseline, thereby ensuring efficiency by design.

- We conduct a comprehensive empirical study across 14 settings: two mathematical reasoning benchmarks (GSM8K, Math500), two base LLMs (Qwen3-32B, LLaMA3-8B) serving either as search policies or BN evaluators, and three representative LITS frameworks (ToT-BS, ReST-MCTS, RAP).

- Our experiments show that **BN-DP consistently reduces runtime by 75–85% across all settings with negligible accuracy loss and in some cases improved accuracy**, confirming its reliability. In contrast, **BN-SC achieves substantial savings in most settings but exhibits instability in a few cases**. Finally, we demonstrate that the overall effectiveness of CiT strongly depends on the accuracy of BN evaluation.

- We release a **unified implementation** of ToT-BS, RAP, and ReST-MCTS with modular LLM-profiled roles, enabling consistent comparison across frameworks and simplifying extension of CiT to future LITS variants.

2 Related Work

We situate our method within a unified view of LLM inference via tree search (LITS), showing how our plug-and-play module can be applied across different frameworks. In addition, we discuss two related lines of work that share a similar high-level motivation—reducing unnecessary branching—but operate in different settings and with distinct mechanisms.

LLM Inference via Tree Search (LITS). A growing line of work has explored performing LLM inference through tree-search (LITS) (Hao et al., 2023; Yao et al., 2023a). Following the unified view of Li (2025), these frameworks can be cast into a Markov Decision Process (MDP), where backbone LLMs are profiled into three main roles: (1) **Policy**: generative LLMs produce candidate actions a_t (e.g., reasoning steps) given a state s_t (e.g., the task with prior steps). This usage traces back to the ReAct paradigm (Yao et al., 2023b); (2) **Reward model**: either using generative LLMs to score candidate steps (Yao et al., 2023a; Hao et al., 2023) or employing dedicated non-generative/fine-tuned models (Dai et al., 2025; Luo et al., 2025); (3) **Transition model**: explicitly predicting the next state s_{t+1} and its confidence r_{conf} when a_t is executed (e.g., RAP (Hao et al., 2023)), or more commonly, updating the state by concatenating new thoughts (e.g., ToT (Yao et al., 2023a), ReST-MCTS (Zhang et al., 2024a)), in which case $r_{\text{conf}} = 1$. Our experiments span both types of reward models and both explicit and concatenation-based transition models, demonstrating that CiT is

broadly compatible across frameworks.

Tree Search Efficiency Broadly, prior work on improving tree-search efficiency for LLMs can be grouped into two orthogonal and complementary directions. The first direction, to which **CiT** belongs, performs *search-level control* by reducing unnecessary branching or reasoning steps. One concurrent line of work is Li et al. (2025), who proposed branching-on-demand in beam search decoding based on entropy over token-level probability distributions. While the high-level motivation is similar—branching only when necessary—the setting is fundamentally different. Their method operates at the token level in traditional sequence generation tasks such as machine translation and speech recognition (Sutskever et al., 2014), where beam search has long been standard. FETCH (Wang et al., 2025a) tackles a related but different decision problem. It assumes that branching has already occurred at the current depth and reduces redundant search by merging semantically similar child nodes via embedding-based clustering. In contrast, CiT decides whether branching should occur in the first place by estimating the Branching Necessity (BN) at the current node, potentially skipping the entire expansion and continuing with a single chained trajectory.

A second direction focuses on improving *intra-step efficiency* under mandatory branching through neural-network-level optimizations, such as speculative execution (Wang et al., 2025b) and prefix-aware tree attention (Yao et al., 2025). These methods primarily target reductions in GPU memory footprint and execution overhead at the expansion step, which is not the focus of the first direction (see Appendix B for details).

3 Chain-in-Tree

We introduce *Chain-in-Tree (CiT)*, a plug-and-play chaining phase inserted before tree expansion, i.e., before actually growing the search tree by attaching k new children at the next depth. [1] CiT adaptively decides whether to grow the tree structure versus to continue in-chain, thereby reducing unnecessary expansions.

We finally analyze the efficiency of *Chain-in-Tree* (CiT) under both Tree-of-Thoughts Beam Search (ToT-BS) and MCTS variants.

[1]While some search procedures (e.g., MCTS rollouts) may involve branching in the sense of simulating multiple continuations, these branches are not materialized in the tree.

3.1 Chaining Phase

As shown in Algorithm 1, during the chaining phase, multiple nodes (actions and their resulting states) are generated and concatenated into a linear chain rather than expanded into branches. This occurs when the BN score r_{bn} exceeds a threshold R_{bn} and the confidence score r_{conf} is below a threshold R_{conf}.

Reusing Children. In standard LLM-in-the-loop tree-search (LITS) frameworks, the expansion phase always invokes the policy to generate a full set of k_{expand} children, regardless of whether a node already has children. CiT modifies this behavior: children generated during chaining are reused at expansion time, truncated to at most k_{expand}, and supplemented with new actions from the policy only if fewer than k_{expand} are available. This design is important to guarantee the efficiency of CiT methods, as specified in Section 3.3and 3.4. Besides, all children—whether reused or newly generated—are consistently assigned rewards, which are not always required during chaining.

Concretely: 1) If $r_{\text{bn}} < R_{\text{bn}}$, the tree is expanded by committing new nodes that contain only the actions produced during chaining; 2) If $r_{\text{bn}} \geq R_{\text{bn}}$ and $r_{\text{conf}} < R_{\text{conf}}$, both actions and their resulting states are retained for expansion.

3.2 Branching Necessity (BN) Evaluation

We now describe how BN scores r_{bn} are computed. We propose two methods: **Direct Prompting (DP)** and **Self-Consistency (SC)**. In both cases, the evaluation operates over k_{bn} candidate actions sampled from the policy model $\text{LLM}_{\text{policy}}$. By default, $k_{\text{bn}} = 1$ for BN-DP and k_{expand} for BN-SC.

Direct Prompting (DP). An auxiliary evaluator model LLM_{bn} is prompted to directly judge whether an action a_t should trigger branching given the current state s_t.

Self-Consistency (SC). A self-consistency strategy (Wang et al., 2023) is adopted by leveraging the diversity of multiple policy samples. From a state s_t, the policy generates a set of k_{bn} candidate actions $\mathcal{A}_t = \{a_1, \ldots, a_{k_{\text{bn}}}\}$. Candidates are clustered into equivalence classes. The BN score r_{bn} is then defined as the fraction of actions belonging to the largest cluster.

$$r_{\text{bn}} = \frac{\max_{C \in \mathcal{C}} |C|}{k_{\text{bn}}}, \tag{1}$$

Algorithm 1 Chain-in-Tree (CiT) with Branching Necessity Evaluation

Input: Depth limit L, BN budget k_{bn}, thresholds $R_{\text{bn}}, R_{\text{conf}}$,
LLM roles: Policy $\text{LLM}_{\text{policy}}$, Transition $\text{LLM}_{\text{trans}}$, Aggregator LLM_{agg}, Equivalence LLM_{eq}, BN evaluator LLM_{bn}
Output: Continued trajectory ending at a node that either triggers branching or is terminal

procedure BN-DP-EVAL(s_t, a_t)
  $r_{\text{bn}} \leftarrow \text{LLM}_{\text{bn}}(s_t, a_t)$
  return r_{bn}
end procedure

procedure BN-SC^1-EVAL($s_t, \mathcal{A}_t$)
  $\mathcal{C} \leftarrow \text{LLM}_{\text{agg}}(\mathcal{A}_t)$ ▷ Cluster candidates
  $r_{\text{bn}} \leftarrow \max_{C \in \mathcal{C}} \frac{|C|}{|\mathcal{A}_t|}$
  return r_{bn}
end procedure

procedure BN-SC^2-EVAL($s_t, \mathcal{A}_t$)
  $\mathcal{P} \leftarrow$ pairs from $\mathcal{A}_t$
  $\mathcal{C} \leftarrow \{\text{LLM}_{\text{eq}}(a_i, a_j) : (a_i, a_j) \in \mathcal{P}\}$ ▷ Equivalence checks
  $r_{\text{bn}} \leftarrow \max_{C \in \mathcal{C}} \frac{|C|}{|\mathcal{A}_t|}$
  return r_{bn}
end procedure

procedure CIT-CHAIN($nd, R_{\text{bn}}, R_{\text{conf}}$)
  initialize path $\leftarrow [\,]$
  while nd is not terminal and $nd.depth \leq L$ **do**
    $\mathcal{A}_t \sim \text{LLM}_{\text{policy}}(nd.state, n = k_{\text{bn}})$
    $r_{\text{bn}} \leftarrow$ BN-Eval$(nd.state, \mathcal{A}_t)$
    append nd to path
    if $r_{\text{bn}} < R_{\text{bn}}$ **then**
      return path ▷ branching required
    end if
    $(s', r_{\text{conf}}) \sim \text{LLM}_{\text{trans}}(nd.state, a)$
    if $r_{\text{conf}} < R_{\text{conf}}$ **then**
      return path ▷ low transition confidence
    end if
    $nd \leftarrow \langle s', a \rangle$
  end while
  append nd to path
  return path
end procedure

where $\mathcal{C}$ = Equivalence classes of $\mathcal{A}_t$.

Intuitively, if most candidates agree on the next step, the model is confident and chaining is applied. If candidates diverge, branching is triggered. By default, chaining occurs when $r_{\text{bn}} \geq 0.5$ (i.e., more than half of the actions are consistent).

Two Implementations of BN-SC We explore two clustering implementations: 1) **BN-SC^1 (Aggregator-based).** An auxiliary model LLM_{agg} clusters candidate actions into dictionaries of the form {canonical action, count}. 2) **BN-SC^2 (Pairwise-Equivalence).** A model LLM_{eq} is queried as a binary oracle to decide whether two candidate actions are semantically equivalent. The union–find–style merging is the applied: each action maintains a representative index, and pairwise equivalences trigger unions. After processing, clusters are defined by representatives and counts are aggregated. In our case, the "representative index" is simply the first surviving index arbitrarily assigned during merging.[2]

In deterministic domains (e.g., board games with fixed move sets), LLM_{eq} can be replaced by simple programmatic rules. Prompts for LLM_{bn}, LLM_{agg}, and LLM_{eq} are detailed in Appendix C.

3.3 Efficiency Guarantee of Beam Search

While LITS frameworks involve multiple LLM roles—policy, reward, and transition models—the cost of policy invocations dominates overall inference and provides the most consistent analytical basis across frameworks. Supporting details are provided in Appendix D, with empirical validation in Section 5.

Assumption (Notation and Setup for ToT-BS (also applies to MCTS))**.** *Let the branching factor be $k_{\text{expand}} \in \mathbb{N}_{>0}$, and let the depth limit be $D \in \mathbb{N}$.*

When a node v is first visited, its BN score *determines whether v is classified as* easy. *If v is easy, the chaining phase is applied: exactly one child is expanded. This consumes k_{bn} policy calls (with $k_{\text{bn}} \leq k_{\text{expand}}$), but only a single child is preserved. Chaining then continues forward until a node fails the BN threshold. At the stopping node (non-easy), a standard full expansion of k_{expand} children is performed, requiring k_{expand} policy calls.*

Expansion cost of ToT-BS. Figure 1 compares the number of policy invocations of Original ToT-BS vs ToT-BS with chaining. Let's assume a homogeneous layer, where every expanded node yields exactlyk_{expand} children (or at least k_{expand} exist) at every depth. At depth t, the frontier size satisfies $\min(B, k_{\text{expand}}^t))$ for all $t \geq 0$. Define the per-depth cost as the number of LLM invocations for action generation, the total expansion cost up to depth D is

$$C_{\text{bs}}(D) = \sum_{t=0}^{D-1} k_{\text{expand}} \min(B, k_{\text{expand}}^t). \quad (2)$$

Guarantee. Assume that D_{C1} is the first depth that normal beam expansion resumes. For

[2] In classical union–find, the root is the unique representative in a hierarchy, while no hierarchy is maintained in our case.

all $t < D_{C1}$ we have $k_{\text{bn}} \leq k_{\text{expand}} \leq k_{\text{expand}} \min(B, k_{\text{expand}}^t) = C_t$, and for all $t \geq D_{C1}$,

$$\min(B, k_{\text{expand}}^{t-D_{C1}}) \leq \min(B, k_{\text{expand}}^t), \quad (3)$$

hence $C_t^{\text{cont}} \leq C_t$. Summing over all depths shows

$$C_{\text{bs+chain}}(D) \leq C_{\text{bs}}(D). \quad (4)$$

Therefore, chaining never increases the number of policy invocations, and yields a strict reduction whenever some $k_{\text{bn}} < k_{\text{expand}}$. The detailed proof is demonstrated in Appendix D.

3.4 Efficiency Guarantee of MCTS

We only consider policy invocation under epxansion. Although there exist policy calls during the simulation phase, chaining shortens the remaining distance to a terminal node before rollout begins, so the simulation phase makes fewer policy calls on average.

Assumption (Notation and Setup for MCTS)**.** *In addition to the notation above, let $N \in \mathbb{N}$ denote the number of MCTS iterations (not fixed). Following prior work (Zhang et al., 2024a; Hao et al., 2023), we adopt the* full expansion on first visit *rule: an unexpanded node generates all k_{expand} actions at once when first expanded.*

Define $E(N)$ as the set of distinct nodes first-visited (i.e., expanded) within N MCTS iterations, and let $E^c(N) \subseteq E(N)$ be the subset of those nodes classified as easy under BN evaluation.

Expansion cost. Under the baseline rule, each new node incurs cost k_{expand}, hence

$$C_{\text{mcts}}(N) = k_{\text{expand}} |E(N)|. \quad (5)$$

With chaining, easy nodes expand with at most k_{bn} calls, while hard nodes expand with k_{expand}, so

$$C_{\text{mcts+chain}}(N) = k_{\text{expand}} \cdot \big(|E(N)| - |E^c(N)|\big) + k_{\text{bn}} \cdot |E^c(N)|. \quad (6)$$

Guarantee. Since $k_{\text{bn}} \leq k_{\text{expand}}$, it follows immediately that

$$C_{\text{mcts+chain}}(N) \leq C_{\text{mcts}}(N), \quad (7)$$

with strict inequality whenever at least one easy node is encountered. Thus adding chaining never increases the number of policy invocations and strictly decreases it in the presence of easy nodes.

4 Experimental Setup

Datasets. Prior work shows that test-time scaling is most effective when models already possess the required background knowledge (Snell et al., 2025). Accordingly, we evaluate on mathematical reasoning, a domain where Chain-of-Thought (CoT) prompting provides substantial improvements (Sprague et al., 2025). We select two standard benchmarks: **GSM8K** and **Math500** (Zhang et al., 2024b; Hendrycks et al., 2021). For strict evaluation, we retain only problems whose final answers are single numbers, yielding 316 usable examples out of the original 500. From both benchmarks, we use the first 100 test instances for evaluation. To verify that results are not driven by subset variance, we additionally evaluate on all 316 filtered Math500 instances; the efficiency–accuracy trends remain consistent (Appendix N).

Baselines. We compare CiT against three representative LLM-in-the-loop tree-search (LITS) frameworks. Each serves as the corresponding upper bound of computational cost for its CiT-augmented variant (details in Appendix E): **(1) ToT-BS** (Yao et al., 2023a): Beam search where $\text{LLM}_{\text{policy}}$ generates candidate thoughts (actions) and LLM_{rm} evaluates candidates. Transitions are realized by concatenation of actions. **(2) RAP** (Hao et al., 2023): MCTS with dynamic transitions and reward models. Each node is defined by a sub-question (action) and its predicted answer (for formulating the next state). We follow the original settings: policy with 4-shot prompting, temperature 0.8 (annealed to 0 at depth limit), generating three candidate branches at each expansion, and 10 MCTS iterations. Rewards are obtained by profiling LLMs for *usefulness*, augmented with an additional correctness check (Appendix F), which we also apply to ToT-BS. $\text{LLM}_{\text{trans}}$ is profiled to answer each sub-question. **(3) ReST-MCTS* (abbreviated as ReST)**: This method is an MCTS variant where each node corresponds to a free-form reasoning step ("thought"). For the *policy*, we use a modified prompt (Appendix C) with temperature 0.7. Other MCTS settings are the same as RAP. For the *transition model*, we follow the same concatenation-based approach as in ToT-BS. For the *evaluator*, we require a Process Reward Model (PRM). The original work introduces a PRM but does not release its trained model. Therefore, we adopt the publicly available PRM from Xiong et al. (2024), which is finetuned on GSM8K

and Math500 training sets, making it a suitable substitute for evaluation in our experiments. **(4) CoT** serves as a lower bound for both efficiency and accuracy.

Base LLMs. Our main experiments use **Qwen3-32B-AWQ**, the strongest open-source model that fits on a single 40GB GPU at the time of writing. We also evaluate with **LLaMA3-8B-Instruct** as a small language model (SLM), consistent with prior findings that SLMs more clearly reveal the benefits of inference-time search (Qi et al., 2025; Zhang et al., 2024a). However, RAP requires a completion-style interface, while chat-formatted models (Qwen3-32B-AWQ, LLaMA3-8B-Instruct) enforce a `(role, content)` structure incompatible with RAP's design. Therefore, we use the completion model **LLaMA3-8B** for RAP. See Appendix G for details. The parameters for LLM inference, including GPU specifications, decoding temperatures, and maximum output token limits, are detailed in Appendix H.

Evaluation Settings. In total, we consider 14 configurations:

- 3 base-LLM settings (Qwen3-32B for all roles; LLaMA3-8B for all roles; LLaMA3-8B for LITS roles with Qwen3-32B for BN roles) $\times$ 2 frameworks (ReST-MCTS, ToT-BS) $\times$ 2 datasets (GSM8K, Math500) = 12 settings.
- Plus RAP with LLaMA3-8B (all roles) on both datasets = 2 additional settings.

CiT does not require a separate BN model. The mixed setting (Qwen3-32B for BN roles only) is included to evaluate the impact of BN evaluator quality.

Metrics. Efficiency is measured by: 1) number of output tokens, 2) number of invocations, 3) wall-clock runtime of $\text{LLM}_{\text{policy}}$, and 4) total runtime. Accuracy is measured using a lightweight number-only evaluator adapted from RAP (Hao et al., 2023), extended with string-to-number normalization and a fallback verification step (via Qwen3-32B) to ensure robustness in parsing numeric answers embedded in text.

Complementary Validation on a Planning Task. We further evaluate CiT on the BlocksWorld benchmark as an initial generalization check beyond mathematical reasoning. Details are provided in Appendix M.

5 Analysis

We report percentage cost savings in Table 1, Table 2 (for ToT-BS and ReST), and Table 3 (for RAP). The complete set of raw measurements underlying these relative improvements are provided in Appendix I.

Overall Efficiency and Stability of BN Methods. Across all settings (or tables), **BN-DP** consistently achieves the efficiency guarantee predicted by our theoretical analysis. In particular, BN-DP reduces total runtime by 75–85% relative to the baselines, while maintaining accuracy comparable to the underlying LITS framework. This makes BN-DP the most reliable and stable choice among our BN evaluation methods.

By contrast, **BN-SC**^{1} performs well in most configurations but fails in 4 out of 14 settings. All failures occur within the ToT-BS framework (**Result Rows 1 & 4** in Table 2): GSM8K and Math500 with both policy and BN roles handled by LLaMA (LLaMA+LLaMA), and GSM8K and Math500 with LLaMA as policy and Qwen as the BN evaluator (LLaMA+Qwen).

BN-SC^2 is more stable, failing in only 1 out of 14 settings, occurring in the ToT-BS framework on Math500 once with LLaMA+Qwen (**Result Row 5** in Table 2). Hence, we recommend BN-DP as the default, empirically stable choice for applying CiT.

Analysis of Failure Settings. To better understand the observed failures, we conducted instance-level analyses for all four failure cases of **BN-SC**^{1} and the failure case of **BN-SC**^{2}. Two consistent patterns emerge: 1) BN methods remain more efficient than the baseline on most instances—for example, on 73% and 83% of Math500 cases with LLaMA+QWen under BN-SC^1 and BN-SC^2, respectively; 2) aggregate inefficiency is driven by the remaining small subset of instances, where BN methods incur more calls than the baseline. This pattern is consistent across all failure settings (Appendix J). We further analyze failure cases by correlating efficiency regressions with reasoning tree structure (Appendix J.2). We find that unusually deep or wide trajectories—rather than problem difficulty—are strong predictors of overhead, suggesting that early detection of such patterns could enable dynamic fallback strategies (e.g., switching to BN-DP). We leave the design of such online mitigation mechanisms to future work.

Method	GSM8K Out	Inv	Time	Total	Acc	Math500 Out	Inv	Time	Total	Acc
Relative to ToT-BS	(baseline Acc: **0.98**; CoT: 0.96)					(baseline Acc: **0.87**; CoT: 0.79)				
+BN-SC^1	13.1%↓	11.8%↓	14.7%↓	16.0%↓	0.96	28.8%↓	2.8%↓	36.7%↓	32.5%↓	0.89
+BN-SC^2	35.2%↓	37.0%↓	38.0%↓	44.4%↓	0.96	62.8%↓	40.3%↓	72.8%↓	71.2%↓	0.84
+BN-DP	78.3%↓	78.3%↓	78.5%↓	77.3%↓	0.97	78.9%↓	80.0%↓	78.1%↓	78.2%↓	0.86
Relative to ReST	(baseline Acc: **0.97**; CoT: 0.96)					(baseline Acc: **0.87**; CoT: 0.79)				
+BN-SC^1	26.7%↓	27.6%↓	34.3%↓	12.1%↑	0.97	41.4%↓	43.7%↓	37.0%↓	25.2%↓	0.84
+BN-SC^2	41.7%↓	44.0%↓	49.0%↓	16.2%↓	0.97	60.7%↓	58.0%↓	58.8%↓	48.7%↓	0.85
+BN-DP	79.6%↓	80.1%↓	81.8%↓	80.3%↓	0.97	82.4%↓	84.9%↓	82.8%↓	82.4%↓	0.88

Table 1: Percentage cost savings relative to ToT-BS and ReST (Qwen3-32B). **In** = input tokens (policy), **Out** = output tokens (policy), **Inv** = number of model invocations (policy), **Time** = wall-clock running time in hours (policy), **Time (Total)** = overall LLM running time in hours (LLMs as policy, reward models, transition models and BN evaluators), **Acc** = accuracy. ↓indicates reductions; ↑indicates overhead.

Method	GSM8K Out	Inv	Time	Total	Acc	Math500 Out	Inv	Time	Total	Acc
Relative to ToT-BS	(baseline Acc: **0.79**; CoT: 0.68)					(baseline Acc: **0.39**; CoT: 0.34)				
BN-SC^1-	14.9%↑	17.4%↑	13.5%↑	21.6%↓	0.71	34.7%↑	37.7%↑	36.7%↑	25.5%↑	0.35
BN-SC^2-	22.4%↓	22.4%↓	20.2%↓	80.9%↓	0.64	38.6%↓	24.8%↓	38.6%↓	76.6%↓	0.30
BN-DP-	68.8%↓	71.0%↓	68.5%↓	73.4%↓	0.73	69.9%↓	63.4%↓	69.7%↓	68.8%↓	0.27
BN-SC^1+	2.9%↓	1.6%↑	1.1%↓	2.3%↑	0.80	36.9%↑	25.5%↑	40.4%↑	62.9%↑	0.38
BN-SC^2+	29.2%↓	29.9%↓	28.1%↓	72.8%↓	0.76	81.2%↑	25.1%↑	87.5%↑	4.9%↓	0.39
BN-DP+	64.0%↓	69.5%↓	62.9%↓	70.9%↓	0.77	37.8%↓	37.8%↓	37.0%↓	39.9%↓	0.36
Relative to ReST	(baseline Acc: **0.80**; CoT: 0.68)					(baseline Acc: **0.39**; CoT: 0.34)				
+BN-SC^1-	22.8%↓	26.0%↓	21.7%↓	35.4%↓	0.70	23.3%↓	17.5%↓	22.4%↓	19.6%↓	0.33
+BN-SC^2-	47.9%↓	47.5%↓	47.2%↓	69.3%↓	0.73	62.8%↓	51.1%↓	63.4%↓	71.0%↓	0.29
+BN-DP-	77.5%↓	77.8%↓	76.4%↓	74.8%↓	0.68	82.4%↓	81.6%↓	82.4%↓	80.6%↓	0.36
+BN-SC^1+	41.0%↓	40.8%↓	40.6%↓	36.1%↓	0.84	65.5%↓	56.7%↓	66.5%↓	50.5%↓	0.43
+BN-SC^2+	50.2%↓	52.1%↓	49.1%↓	67.6%↓	0.80	39.8%↓	48.4%↓	40.9%↓	53.8%↓	0.34
+BN-DP+	78.8%↓	80.6%↓	78.3%↓	76.3%↓	0.76	66.7%↓	67.7%↓	67.1%↓	65.0%↓	0.37

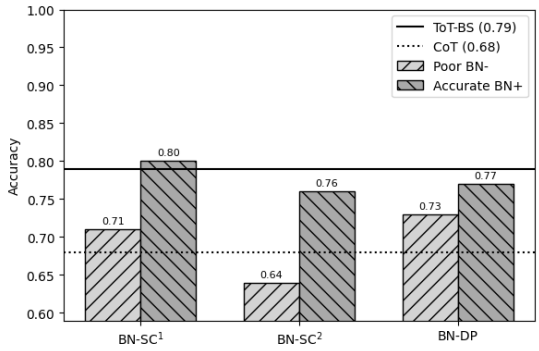
Table 2: Percentage cost savings relative to ToT-BS and ReST (LLaMA3-8B Instruct), under both *Poor BN* evaluation (–, using LLaMA-3-8B Instruct) and *Accurate BN* evaluation (+, using Qwen-3-32B).

Method	GSM8K Out	Inv	Time	Total	Acc	Math500 Out	Inv	Time	Total	Acc
Relative to RAP	(baseline Acc: **0.61**; CoT: 0.32)					(baseline Acc: **0.18**; CoT: 0.17)				
BN-SC^1+	65.3%↓	69.9%↓	65.2%↓	47.9%↓	0.48	19.5%↓	34.5%↓	9.0%↓	2.4%↓	0.27
BN-SC^2+	78.0%↓	75.5%↓	78.3%↓	80.8%↓	0.46	63.4%↓	59.6%↓	61.2%↓	60.4%↓	0.21
BN-DP+	88.6%↓	86.4%↓	88.4%↓	77.4%↓	0.57	79.1%↓	80.5%↓	79.1%↓	60.8%↓	0.26

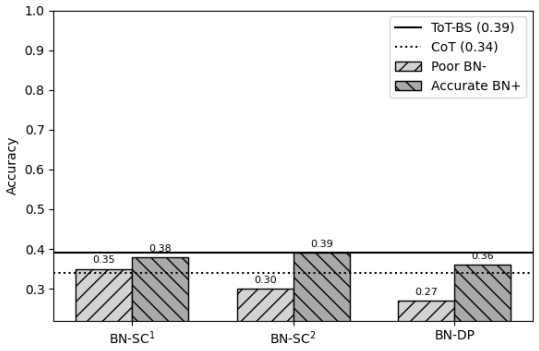
Table 3: Percentage cost savings relative to RAP (LLaMA3-8B), where *Accurate BN* evaluation (+) is performed by using Qwen-3-32B for BN roles.

Impact of BN Evaluator Quality on Accuracy. Figure 2 illustrates that the quality of BN evaluation plays a critical role in the effectiveness of our CiT plug-in. When smaller LLMs (e.g., LLaMA-3-8B) are used as BN evaluators (*Poor BN*), the accuracy often drops below that of the baseline LITS methods and in some cases approaches the weaker CoT baseline. This suggests that underpowered BN evaluators may systematically overestimate node scores, leading to premature chaining and insufficient exploration, which propagates errors downstream.

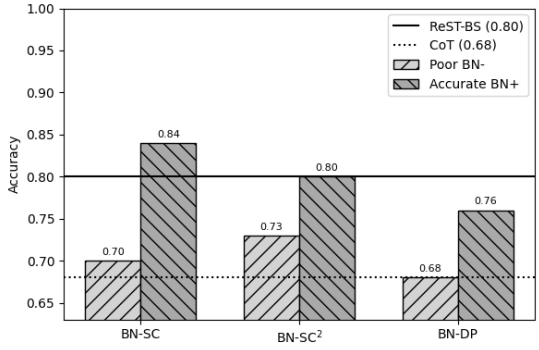
In contrast, when stronger LLMs (e.g., Qwen-3-32B) are used as BN evaluators (*Accurate BN*), the performance recovers and in most cases approaches that of the baseline LITS methods. While in some settings with BN-SC^1 performance can even appear higher than baseline LITS, we caution against overinterpreting this result: it may partly reflect the ability of the BN aggregator to reformulate or im-



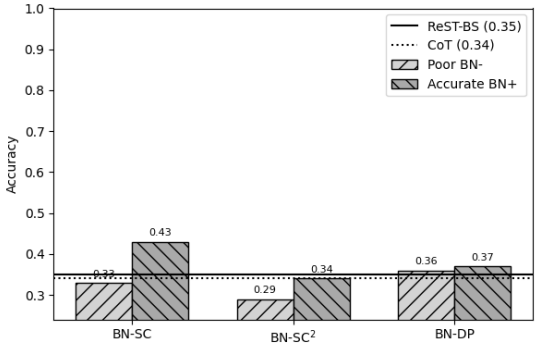
(a) ToT-BS + CiT on GSM8K



(b) ToT-BS + CiT on Math500



(c) ReST-MCTS* + CiT on GSM8K



(d) ReST-MCTS* + CiT on Math500

Figure 2: Accuracy comparison under CiT plug-in across two search frameworks (ReST-MCTS* and ToT-BS) and two datasets (GSM8K, Math500). Bars show BN evaluator quality (**Poor BN**: LLaMA-3-8B vs **Accurate BN**: Qwen-3-32B). Horizontal lines denote baselines (CoT, ReST, ToT-BS).

prove upon actions generated by the policy rather than simply. Thus, we do not claim that BN-SC can outperform baseline LITS.

The consistent finding across both frameworks (ReST-MCTS* and ToT-BS) is that chaining can be effective when BN evaluation is sufficiently accurate, while poor BN evaluation can negate the benefits of chaining or even harm performance. A reasonable implication is that, for tasks with deterministic action spaces, reliable BN evaluation could in principle be implemented with rule-based or hard-coded checks, thereby eliminating the risks associated with poor BN evaluators.

Additional Planning Evaluation (BlocksWorld). On the BlocksWorld planning task, CiT exhibits efficiency improvements consistent with those observed on mathematical reasoning benchmarks. When equipped with BN-DP, CiT preserves a substantial portion of RAP's planning performance while reducing inference cost, whereas BN-SC achieves larger efficiency gains at the expense of solution quality by aggressively suppressing branching. Full results are reported in Appendix M.

6 Conclusion

We proposed **Chain-in-Tree (CiT)**, a plug-and-play framework that inserts a chaining phase into LLM tree search to avoid unnecessary branching. CiT theoretically guarantees non-increasing policy cost and achieves up to 85% runtime reduction across ToT-BS, ReST-MCTS, and RAP without accuracy loss. Overall, CiT emphasizes the importance of accurate BN evaluation for scaling LLM-based search.

6.1 Reproducibility

We release an open-source Python package that modularizes LLM-profiled roles across search frameworks and provides scripts to reproduce all experiments. The CiT chaining phase is implemented as a single framework-agnostic function. Implementation details are provided in Appendix L.

For transparency, the supplementary material includes datasets, execution logs, search-tree reconstructions, and per-instance inference cost reports.

Limitations

Coverage of LITS Frameworks. Our study focuses on heuristic-light LITS settings where LLMs drive search decisions.Other search paradigms such as A* or heuristic-guided search are not evaluated. Nevertheless, our use of unified tasks and LLM-profiled roles (Li, 2025), together with modularized implementations, makes it straightforward to extend CiT to additional LITS frameworks in future work.

Scope of Empirical Evaluation. Following prior work (Snell et al., 2025; Zhang et al., 2024a), our main experiments focus on mathematical reasoning, where LLMs already possess the necessary knowledge but still benefit from test-time scaling. We include BlocksWorld (Appendix M) as an initial evaluation on a planning domain with deterministic action spaces, but broader coverage of such domains (e.g., navigation, other board games) where BN-SC's use of LLMs for clustering semantically equivalent actions may be unnecessary remains future work.

Accuracy Gains from CiT. Although CiT is designed for efficiency, we observe that it can also improve accuracy in several settings. Understanding why chaining sometimes enhances reasoning quality requires deeper analysis, which we leave for future investigation.

References

Ziru Chen, Michael White, Ray Mooney, Ali Payani, Yu Su, and Huan Sun. 2024. When is tree search useful for LLM planning? it depends on the discriminator. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13659–13678, Bangkok, Thailand. Association for Computational Linguistics.

Ning Dai, Zheng Wu, Renjie Zheng, Ziyun Wei, Wenlei Shi, Xing Jin, Guanlin Liu, Chen Dun, Liang Huang, and Lin Yan. 2025. Process supervision-guided policy optimization for code generation.

Arthur Guez, David Silver, and Peter Dayan. 2012. Efficient bayes-adaptive reinforcement learning using sample-based search. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 1025–1033, Red Hook, NY, USA. Curran Associates Inc.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8154–8173, Singapore. Association for Computational Linguistics.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Xianzhi Li, Ethan Callanan, Xiaodan Zhu, Mathieu Sibue, Antony Papadimitriou, Mahmoud Mahfouz, Zhiqiang Ma, and Xiaomo Liu. 2025. Entropy-aware branching for improved mathematical reasoning. *arXiv preprint arXiv:2503.21961*.

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

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Meiqi Guo, Harsh Lara, Yunxuan Li, Lei Shu, Lei Meng, Jiao Sun, and Abhinav Rastogi. 2025. Improve mathematical reasoning in language models with automated process supervision.

Zhenting Qi, Mingyuan Ma, Jiahang Xu, Li Lyna Zhang, Fan Yang, and Mao Yang. 2025. Mutual reasoning makes smaller LLMs stronger problem-solver. In *The Thirteenth International Conference on Learning Representations*.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2025. Scaling test-time compute optimally can be more effective than scaling LLM parameters. In *The Thirteenth International Conference on Learning Representations*.

Zayne Rea Sprague, Fangcong Yin, Juan Diego Rodriguez, Dongwei Jiang, Manya Wadhwa, Prasann Singhal, Xinyu Zhao, Xi Ye, Kyle Mahowald, and Greg Durrett. 2025. To cot or not to cot? chain-of-thought helps mainly on math and symbolic reasoning. In *The Thirteenth International Conference on Learning Representations*.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27.

Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2023. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Ante Wang, Linfeng Song, Ye Tian, Dian Yu, Haitao Mi, Xiangyu Duan, Zhaopeng Tu, Jinsong Su, and Dong Yu. 2025a. Don't get lost in the trees: Streamlining LLM reasoning by overcoming tree search

exploration pitfalls. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23946–23959, Vienna, Austria. Association for Computational Linguistics.

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

Zhenglin Wang, Jialong Wu, Yilong Lai, Congzhi Zhang, and Deyu Zhou. 2025b. SEED: Accelerating reasoning tree construction via scheduled speculative decoding. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 4920–4937, Abu Dhabi, UAE. Association for Computational Linguistics.

Wei Xiong, Hanning Zhang, Nan Jiang, and Tong Zhang. 2024. An implementation of generative prm. https://github.com/RLHFlow/RLHF-Reward-Modeling.

Jinwei Yao, Kaiqi Chen, Kexun Zhang, Jiaxuan You, Binhang Yuan, Zeke Wang, and Tao Lin. 2025. DeFT: Decoding with flash tree-attention for efficient tree-structured LLM inference. In *The Thirteenth International Conference on Learning Representations*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. *Preprint*, arXiv:2305.10601.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023b. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. 2024a. ReST-MCTS*: LLM self-training via process reward guided tree search. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Di Zhang, Xiaoshui Huang, Dongzhan Zhou, Yuqiang Li, and Wanli Ouyang. 2024b. Accessing gpt-4 level mathematical olympiad solutions via monte carlo tree self-refine with llama-3 8b. *arXiv preprint arXiv:2406.07394*.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2024. Language agent tree search unifies reasoning acting and planning in language models.

A Algorithmic Design of Search Control

Search Control with CiT. CiT introduces a new chaining phase that intervenes before node expansion, altering the control flow of tree search by deciding whether structural branching should occur at all. This requires nontrivial algorithmic changes, including: (i) reusing partially generated children across chaining and expansion phases; (ii) decoupling action generation from reward assignment and state expansion so to save unnecessary inference cost for chaining nodes; (iii) hence, during the expansion phase, additional checking is required to ensure rewards for nodes generated during chaining phase but not identified for chaining iiii) identifying chaining nodes and treating them correctly within iterative processes and across other search phases, including an additional check before UCT selection. These design choices are essential for making a seemingly simple principle work uniformly across ToT-BS, ReST-MCTS, and RAP, and are fundamentally different from prior approaches that operate after expansion (e.g., node merging) or at the token level (e.g., speculative decoding).

Task-Agnostic Abstraction for Tree Search. We generalize the task abstraction to support BN evaluation and chaining decisions that are independent of task formulation (e.g., although action is more direct abstraction for policy. abstracting it as a part of Step can streamline the information flow from Policy to Transition and Reward Models. The task-specific attributes beyond actions, such as observation, state snapshots in BlocksWorld, but are necessary for transition and reward models can be instantiated) and framework-specific prompt structures. This design is essential for making CiT truly plug-and-play across different frameworks, and goes beyond prior work that tightly couples search logic to task-specific representations. Concretely, we unify the original query in language reasoning tasks and the evolving goal into a single state in environment-grounded tasks (e.g., BlocksWorld). Such abstraction can be uniformly consumed by the search procedure with different policy, reward models, and transition model.

B Cost and Resource Analysis

We evaluate efficiency primarily using token counts (input and output), model invocations, and wall-clock runtime. Below we clarify how these metrics relate to other commonly discussed cost measures.

FLOPs. For decoder-only Transformer architectures with fixed model size and implementation, the dominant computational cost at inference arises from forward passes over input prefixes and generated tokens. As a result, the total number of input

and output tokens provides a close proxy for total FLOPs, up to constant factors determined by the architecture and hardware. This practice is standard in LLM-in-the-loop tree search (LITS) and test-time compute (TTC) literature (e.g., ReST-MCTS*), where token usage and runtime are used to characterize computational efficiency. Consequently, we do not separately report FLOPs.

Memory Footprint. All methods compared in this work use the same base model and run on identical hardware configurations. CiT does not modify the underlying model architecture, context length, or execution graph. Therefore, peak GPU memory usage remains effectively constant across baselines and CiT variants in typical (non-low-resource) settings. Accurately measuring fine-grained GPU memory dynamics would require system- and implementation-specific profiling, which is orthogonal to our algorithmic focus and omitted here.

API Cost. Our experiments are conducted using local open-weight models. For commercial LLM APIs, pricing is typically defined as a linear function of the number of input and output tokens processed. Under such pricing models, the token reductions achieved by CiT translate directly into proportional reductions in API cost. As API pricing is provider-specific, we do not report absolute monetary costs.

C Prompts

The prompt use as policy, reward model and world model is summarized in Table 4.

Methods	Policy	Reward Model	Dynamic Model
RAP	QA-based policy (Table 5)	Usefulness logit-based evaluator (Table 7)	QA-based transition model (Table 5)
ReST-MCTS*	Thought generator (Table 6)	Fine-tuned reward models	NA
ToT-BF	Thought generator (Table 6)	Usefulness evaluator(Table 9) + Correctness evaluator (Table 8)	NA

Table 4: Prompt Summary for Different Frameworks.

Given a question, please decompose it into sub-questions. For each sub-question, please answer it in a complete sentence, ending with "The answer is". When the original question is answerable, please start the subquestion with "Now we can answer the question:"
(few-shot demonstrations are ignored)
Question 1: James writes a 3-page letter to 2 different friends twice a week. How many pages does he write a year?
Question 1.1: How many pages does he write every week?
Answer 1.1: James writes a 3-page letter to 2 different friends twice a week, so he writes 3 * 2 * 2 = 12 pages every week. The answer is 12.
Question 1.2: How many weeks are there in a year?
Answer 1.2: There are 52 weeks in a year. The answer is 52.
Question 1.3: Now we can answer the question: How many pages does he write a year?
Answer 1.3: James writes 12 pages every week, so he writes 12 * 52 = 624 pages a year. The answer is 624.
...

Question 5: Janet's ducks lay 16 eggs per day. She eats three for breakfast every morning and bakes muffins for her friends every day with four. She sells the remainder at the farmers' market daily for $2 per fresh duck egg. How much in dollars does she make every day at the farmers' market?
Question 5.1:

(Generation as a policy)
How many eggs does Janet have left after eating three for breakfast and baking muffins with four?

Answer 5.1:

(Generation as a transition model)
Janet starts with 16 eggs per day, consumes 3 for breakfast and 4 for baking, so she has 16 - 3 - 4 = 9 eggs left for the farmers' market. The answer is 9.

Table 5: A policy/transition model for RAP. Sourced from Hao et al. (2023).

System message:

Your task is to give the correct next step, given a science problem and an existing partial solution (not a complete answer).
Assuming the input is n-steps, then the format of the input is:
"Problem: ...
Existing Steps:
Step 1: ...
Step 2: ...
...
Step n: ..."
where ... denotes omitted input information.

Please follow the restricted output format:
* If no existing steps are given, generate Step 1.
* Otherwise, following the given step(s), output ONLY ONE step within 1000 tokens. SHOULD NOT output multiple steps.
* DO NOT repeat Problem or any Existing Steps.
* Your output should be a complete reasoning step that includes calculations, reasoning, choosing answers, etc.
* When the final answer is/has been reached, begin the step with EXACTLY the phrase:"Now we can answer the question: The answer is ", followed by EXACTLY one number. Do not include any other words, punctuation, or explanation after the number.

User message:

Problem: How many positive whole-number divisors does 196 have?
Existing Steps: None
Step 1:

Table 6: A policy used for ReST and BFS. Sourced from Zhang et al. (2024a).

Given a question and some sub-questions, determine whether the last sub-question is useful to answer the question. Output 'Yes' or 'No', and a reason.
Question 1: Four years ago, Kody was only half as old as Mohamed. If Mohamed is currently twice as 30 years old, how old is Kody?
Question 1.1: How old is Mohamed?
Question 1.2: How old was Mohamed four years ago?
New question 1.3: How old was Kody four years ago?
Is the new question useful? Yes. We need the answer to calculate how old is Kody now.
(Few-shot demonstrations are ignored.)

Question 5: Janet's ducks lay 16 eggs per day. She eats three for breakfast every morning and bakes muffins for her friends every day with four. She sells the remainder at the farmers' market daily for $2 per fresh duck egg. How much in dollars does she make every day at the farmers' market?
New question 5.1: Now we can answer the question: How much in dollars does she make every day at the farmers' market?
Is the new question useful?

Table 7: An LLM evaluator. Source from Hao et al. (2023)

System message:

Given a question and a chain of thoughts, determine whether the last thought is **correct**, where **correct** means factually accurate and mathematically accurate (all calculations and formulas are correct), and logically consistent with the question.
Instructions:
Output only a score:
- 0 if any correctness criterion is unmet.
- 1 if all correctness criteria are fully met.
The score must be parsable by Python's `float()` function, with no punctuation or additional text.

User message:

Problem: How many positive whole-number divisors does 196 have?
Existing Steps:
Step 1: Find the prime factorization of 196 as $2^2 \times 7^2$
Step 2: Apply the divisor formula $(a+1)(b+1)$ to the prime factorization $2^2 \times 7^2$ and calculate $3 \times 3 = 9$
New Step to be evaluated: Now we can answer the question: The answer is 9.

Table 8: A correctness evaluator used for BFS.

System message:

Given a question and a chain of thoughts, determine how **useful** the last thought is for answering the question, regardless of correctness.
Instructions:
Output only a score between 0 and 1:
- 0 if the step is entirely irrelevant or unhelpful.
- 1 if the step is essential and maximally useful.
- A value strictly between 0 and 1 if the step is partially useful. Larger values indicate more usefulness.
The score must be parsable by Python's `float()` function, with no punctuation or additional text.

User message:

Problem: How many positive whole-number divisors does 196 have?
Existing Steps:
Step 1: Find the prime factorization of 196 as $2^2 \times 7^2$
Step 2: Apply the divisor formula $(a+1)(b+1)$ to the prime factorization $2^2 \times 7^2$ and calculate $3 \times 3 = 9$
New Step to be evaluated: Now we can answer the question: The answer is 9.

Table 9: A usefulness evaluator used for BFS.

System message:

You are an expert at deciding whether a single reasoning step is *logically compulsory* given the task and the partial solution path.
Input fields (A) Task description - one paragraph.
(B) Partial reasoning path so far.
(C) Candidate next step.
ONLY output a single number from 1 to 4.
Scale
4 - **Unavoidable next step**: given the current path, this step must come next to proceed logically.
3 - Strongly expected: skipping it now would be very unusual, though not impossible.
2 - Potentially useful but avoidable: alternative coherent next steps exist.
1 - **Optional**: the step is not logically required at this point.
Think silently, then output the single line - nothing else.

User message:

(A) *(task)*
(B) *(partial path)*
(C) *(candidate step)*

Table 10: BN Evaluator.

System message for RAP:

You are given a QUESTION and its partial solution (Subquestions which have been answered).
Your task is to group the provided list of candidate next subquestions (After "List of Candidates for the following step") into clusters.

- Steps that are semantically equivalent must be grouped together.
- Paraphrase or stylistic differences are irrelevant
- Existing Steps are given only as context and MUST NOT appear in the clusters.

OUTPUT FORMAT (Python literal list only; must be parsable by ast.literal_eval):
OUTPUT FORMAT:

```
[
 { "canonical_action": "<a CONCRETE subquestion>", "count": <the number of the candidates grouped in that cluster> },
 ...
]
```

Rules:
- Each array element represents one cluster.
- No text outside the list.
- The total number of generated words should be NO more than 450 words.

System message for ReST-MCTS* and ToT-BS:

You are given a QUESTION and its partial solution (Existing Steps).
Your task is to group the provided list of candidate next steps (After "List of Candidates for the following step") into clusters.

- Steps that are semantically equivalent must be grouped together.
- Paraphrase or stylistic differences are irrelevant.
- Existing Steps are given only as context and MUST NOT appear in the clusters.

OUTPUT FORMAT:

```
[
 { "canonical_action": "<CONCRETE calculation(s) and outcome(s) after the Existing Steps>", "count": <the number of the candidates grouped in that cluster> },
 ...
]
```

Rules: Each array element represents one cluster. No text outside the list. The total number of generated words should be no more than 450 words.

Table 11: BN Aggregator .

System message for RAP:
You are a strict semantic comparator.
Given two sub-questions, decide if they are semantically overlapping given the context.

System message for ReST-MCTS* and ToT-BS:
You are a strict semantic comparator.
Given two action descriptions, decide if they are semantically overlapping given the context.
Definition:
- "Overlapping" means the two descriptions express the same underlying operation or one is a specific case/subsumption of the other or have the same effect on the context.
- "Not overlapping" means the operations are mutually exclusive in meaning.
Answer format: return only 'YES' or 'NO' with no punctuation, no explanation.

User message:
Context:
========
{*context*}
========
New Step A: {*canonical_action*}
New Step B: {*canonical_action*}
Do these steps express the same underlying operation given the context?

Table 12: BN equivalence checker

D Theoretical Costs

D.1 Scope of Theoretical Analysis

This section clarifies why our theoretical efficiency analysis of *Chain-in-Tree* (CiT) focuses exclusively on policy invocations.

Justification for Focusing on Policy Invocations. 1) Cost of reward and transition models never increase: While LLMs may serve as *policy*, *reward*, and *transition* models, we only count invocations of $\text{LLM}_{\text{policy}}$. The reason is structural: every node generated by $\text{LLM}_{\text{policy}}$ necessarily triggers the use of LLM_{rm} and/or $\text{LLM}_{\text{trans}}$ if it exists. For example, in ToT-BS, every child node requires reward evaluation; in RAP, at least one transition evaluation is needed per depth to maintain expansion. By contrast, during the chaining phase, CiT invokes $\text{LLM}_{\text{trans}}$ exactly once per chaining node (i.e., per depth), which is no worse than the per-depth transition usage in the original frameworks; moreover, reward evaluation via LLM_{rm} can be deferred until branching. Thus, CiT consistently reduces reward/-transition overhead under the same policy budget.

(2) Additional BN evaluation roles are lightweight. Although CiT introduces additional roles—LLM_{bn}, LLM_{agg}, and LLM_{eq}—for Branching Necessity (BN) evaluation, their overhead is significantly smaller than that of $\text{LLM}_{\text{policy}}$ or $\text{LLM}_{\text{trans}}$ (when present). We confirm this empirically in Section 5.

Uniform Token Length Assumption. We assume the number of input and output tokens per $\text{LLM}_{\text{policy}}$ invocation is approximately uniform across steps and examples.

D.2 ToT-BS

Proposition 1. (Beam-search frontier size.) At depth t, the frontier size under beam search satisfies

$$|N_t| = \min(B, k_{\text{expand}}^t), \tag{8}$$

assuming each node generates at most k_{expand} children, pruning retains the top B nodes at each layer, and no terminals are reached.

Proof. The recurrence relation for the frontier is

$$|N_{t+1}| = \min\big(B,\ k_{\text{expand}}\,|N_t|\big). \tag{9}$$

For the base case, $|N_0| = 1 = \min(B, k_{\text{expand}}^0)$ (assuming $B \geq 1$). For the inductive step, assume $|N_t| = \min(B, k_{\text{expand}}^t)$. Then

$$\begin{aligned} |N_{t+1}| &= \min\big(B,\ k_{\text{expand}} \min(B, k_{\text{expand}}^t)\big) \\ &= \min\big(B,\ \min(k_{\text{expand}}B,\ k_{\text{expand}}^{t+1})\big). \end{aligned} \tag{10}$$

Because $k_{\text{expand}}B \geq B$ for all integers $k_{\text{expand}} \geq 1$, the inner minimum simplifies to k_{expand}^{t+1} whenever $k_{\text{expand}}^{t+1} < B$, and otherwise the outer minimum enforces B. Thus

$$|N_{t+1}| = \min(B, k_{\text{expand}}^{t+1}).$$

□

Hence the frontier grows geometrically as k_{expand}^t until saturating at beam width B. An edge case is $k_{\text{expand}} = 1$, which yields a single chain $|N_t| = 1$ (if $B \geq 1$).

In the presence of terminals or variable branching, the relation holds as an upper bound rather than an equality, i.e., $|N_t| \leq \min(B, k_{\text{expand}}^t)$.

Proposition 2. (Beam-search cumulative expansion cost.) Assume each node generates at most k_{expand} children, pruning retains the top B nodes after each layer, and no terminal states are encountered. Let $D \in \mathbb{N}$ be the maximum depth (number of layers). Define the per-depth expansion cost as the number of child-generation operations,

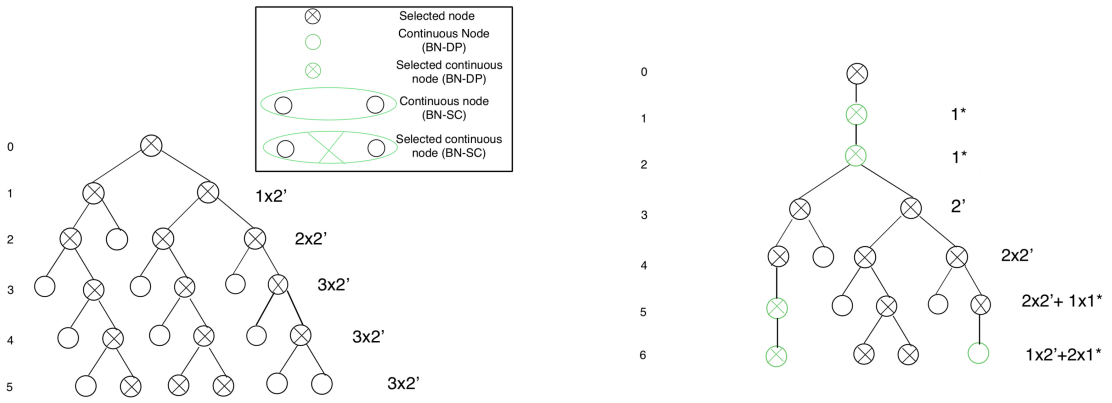
$$\begin{aligned} C_{\text{bs}}(t) &:= k_{\text{expand}}\,|N_t| \\ &= k_{\text{expand}}\ \min(B,\ k_{\text{expand}}^t), \\ &\quad t = 0, 1, \ldots, D-1. \end{aligned} \tag{11}$$

Then the total expansion cost up to depth D is

$$\begin{aligned} C_{\text{bs}}(D) &= \sum_{t=0}^{D-1} C_t \\ &= \sum_{t=0}^{D-1} k_{\text{expand}}\ \min(B,\ k_{\text{expand}}^t). \end{aligned} \tag{12}$$

Proposition 3. (Beam-search with chaining: cumulative expansion cost.) Suppose chaining is enabled for the first D_{C1} "easy" depths, where each expansion produces at most $k_{\text{bn}} \leq k_{\text{expand}}$ children ($k_{\text{bn}} = 1$ for direct prompting, $k_{\text{bn}}^t = k_{\text{expand}}$ for the self-consistency approach). After depth D_{C1}, standard beam search resumes with branching factor k_{expand} and beam size B. Then the per-depth expansion cost is

Hence the total expansion cost up to depth D is



(a) ToT-Beam Search.

(b) ToT-Beam Search + Chaining (BN-DP).

Figure 3: The number of policy invocations of Original ToT-BS vs ToT-BS + Chaining. The numbers suffixed by "*l*" indicate sampling size for beam search expansion, while The numbers suffixed by "*" indicate sampling size for BN judge.

$$
\begin{aligned}
&C_{\text{bs+chain}}(D) \\
&= \sum_{t=0}^{D_{C1}-1} k_{\text{bn}} \\
&\quad + \sum_{t=D_{C1}}^{D-1} k_{\text{expand}} \cdot \min\left(B,\, k_{\text{expand}}^{t-D_{C1}}\right).
\end{aligned}
\tag{13}
$$

□

D.3 Theoretical Error Analysis of BN Evaluation

BN only affects search through the *chaining decision*. At a node with state s_t, BN evaluation produces a score r_{bn} and chaining is applied when $r_{\text{bn}} \geq R_{\text{bn}}$ (and, when applicable, $r_{\text{conf}} < R_{\text{conf}}$). Therefore, BN errors can only change the search procedure by changing *how often* nodes are classified as *necessary*, i.e., how frequently chaining is used instead of standard expansion.

ToT-BS: BN errors shift the realized D_{C1}. Recall that D_{C1} is defined as the first depth at which normal beam expansion resumes. If BN is optimistic (branching is suppressed), more depths are classified as necessary and chaining persists longer, which increases the realized D_{C1}. After depth D_{C1}, the frontier size follows the same form as in Appendix D.2:

$$
|N_t| = \min\left(B,\, k_{\text{expand}}^{t-D_{C1}}\right), \qquad t \geq D_{C1},
$$

so a larger D_{C1} reduces the amount of branching available at a fixed depth t and makes the search trajectory more chain-like. If BN is pessimistic (branching is over-triggered), chaining stops earlier (smaller D_{C1}) and the procedure approaches the original ToT-BS.

Crucially, regardless of BN bias, the policy-invocation cost never increases: as shown in Section 3.3,

$$
C_{\text{bs+chain}}(D) \;\leq\; C_{\text{bs}}(D),
$$

with strict reduction whenever some necessary depths use $k_{\text{bn}} < k_{\text{expand}}$. Thus, overly optimistic BN may reduce exploration breadth (and potentially accuracy), but it does not harm efficiency relative to the baseline.

MCTS: BN errors change which first-expanded nodes are cheap. Under the *full expansion on first visit* rule (Section 3.4), the set $E(N)$ denotes the distinct nodes that are first-expanded within N MCTS iterations, and $E^c(N) \subseteq E(N)$ denotes those among them classified as necessary by BN. BN affects the expansion cost only through whether a first-expanded node uses k_{bn} (necessary) or k_{expand} (unnecessary), as shown in Equations 5 and 6. Optimistic BN increases $|E^c(N)|$ (more first-expanded nodes are treated as necessary), which pushes the explored tree toward cheaper, narrower expansions. Pessimistic BN decreases $|E^c(N)|$, which reduces chaining opportunities and makes the procedure closer to baseline MCTS.

Note that $|E(N)| \leq N$ always holds, and $|E(N)|$ can be strictly smaller than N when iterations frequently terminate early (e.g., reaching

terminal states) or hit the depth limit D before encountering a new unexpanded node. This phenomenon is independent of BN and does not affect the monotonic guarantee below.

Guarantee under BN error. For both ToT-BS and MCTS, the efficiency guarantees hold regardless of BN bias because the design enforces $k_{\text{bn}} \leq k_{\text{expand}}$. In particular, for MCTS we always have

$$C_{\text{mcts+chain}}(N) \;\leq\; C_{\text{mcts}}(N),$$

with strict inequality whenever at least one "'necessary" node is encountered (i.e., $|E^c(N)| > 0$). Therefore, BN errors primarily trade off *exploration breadth* versus *cost savings*, while never increasing the number of policy invocations relative to the baseline.

Takeaway. Using only existing quantities, BN evaluation errors map to downstream search behavior through (i) the realized D_{C1} in ToT-BS and (ii) the realized subset $E^c(N)$ of first-expanded nodes in MCTS. A full theory that further maps these structural changes to *solution accuracy* would require modeling how reduced branching affects the probability of encountering a correct trajectory within depth limit D (or within N iterations), which we leave as future work and instead study empirically in Section 5 and Appendix M.

E Details of RAP and REST-MCTS

E.1 Task Formulation: RAP vs ReST-MCTS

Table 13 shows the distinction between task formulation.

E.2 Four Phases

As standard MCTS, four phases are included for RAP and REST-MCTS.

- Selection: Walk down the tree until a leaf or an unexpanded node is reached. UCT values will be used for selection when all the child nodes have been visited (i.e., node.state is not empty). Otherwise , the reward is estimated to select from unvisited nodes.

- Expansion: If the leaf is not terminal and not depth-limited, the selected node will be expaned by calling the policy. All its k children will be expanded at once.
  - Lazy Sampling: Following RAP implementation (Hao et al., 2023), lazy sampling (Guez et al., 2012) is used. After the policy generates multiple actions, RAP may leave action nodes unvisited for efficiency and only use the transition model to infer the next state if it is selected for expansion. Therefore, the sampled candidate nodes only maintain actions.
  - Non-empty state may reflect the existence of value: Once the state of a node is not empty, the node must have gone through backpropagation and thus contain rewards for selection. But this does not be the case for continuous nodes due to the requirement of inferring the state during continuation.

- Simulation (rollout): At each step, k actions are sampled from policy, and then most reward models are used to select the most valuable one.
  - In RAP, the sampling process will proceed iteratively until a terminal step is reached or a global depth limit is reached.
  - In ReST-MCTS, a fixed depth limit 2 is set specifically for roll-out.

- MCTS Backpropagation - Value Update: After each simulation returns a reward r, update the Q value as:

$$Q_{\text{new}} = \frac{r + Q_{\text{old}} \cdot \text{Count}_{\text{new}}}{\text{Count}_{\text{new}}}, \qquad (14)$$

  r, depending on the task, can be a reward at the terminal state. In some cases,it can be an aggregated one, if each simulation step yields a reward. Specifically, if rewards r_t are discounted by γ, then the final sample reward r for backpropagation is:

$$r = G = \sum_{t=0}^{T-1} \gamma^t r_t. \qquad (15)$$

  These rewards from simulated nodes are then employed to update the Q values for non-simulated nodes, including the leaf nodes and those above. Q_{old} are the previous Q-value, and $\text{Count}_{\text{new}}$ is the total visit count after the current update.

Reasoning via Concatenation from Rest-MCTS	Reasoning via QA from RAP
First, calculate the total number of eggs used by Janet each day for breakfast and baking.(Homogeneous Thought 1) Janet uses 3 eggs for breakfast and 4 eggs for baking, so the total number of eggs used each day is $3 + 4 = 7$. (Homogeneous Thought 2)	What is the total number of eggs used by Janet each day for breakfast and baking? (Question as action) Janet uses 3 eggs for breakfast and 4 eggs for baking, so the total number of eggs used each day is $3 + 4 = 7$. (Answer to form the next state)

Table 13: Formatting Difference Between Reasoning via Concatenation and QA under the same meaning.

During our implementation, each reward $r \in R_{\text{cum}}$ propagating from the terminal node is stored in a list R_{cum} , and average them when used. The procedure of value estimate, provides the initialized values for new actions or resulting states.

For RAP, the confidence of transition models is also included along with the usefulness score to be processed for backpropagation

$$r^w \cdot r_{\text{conf}}^{(1-w)}, \tag{16}$$

where w is the weight between 0 and 1.

- MCTS Backpropagation - Visit Update: Except for the estimated value $Q(s, a)$, the backpropagation also updates the visit count for every state on the path from the root to the leaf and each edge (s, a) along that path, denoted as Count (s_t) and Count (s_t, a_{t+1}), respectively.

F Reward Models

We profile LLMs as reward models, as RAP does. However, the original profiling only consider the contribution of solution steps to the input task. As suggested by Zhang et al. (2024a), we further improve the prompt to reflect the probability of correctness. Three constraints are enforced: bounded range, contribution-awareness, and correctness-awareness in the original paper.

G Incompatibility of Chat-Formatted LLMs with RAP

Firstly, the policy and transition model in RAP requires a raw completion interface: the LLM is repeatedly invoked with concatenated sub-questions and answers, such as "`Subquestion 1: ... Answer 1: ... Subquestion 2: ...`". This mismatch often leads to unstable generations (e.g., empty outputs, spurious punctuation, or off-task continuations), since the evolving transcript no longer resembles the conversational templates seen in training. Furthermore, the reward model requires the next token logically requiring "`yes`" or "`no`" for judgments (e.g., "`Is the new question useful?`"), however, the next token in the chat models tends to be a template one, which disrupts the intended binary evaluation.

H Parameters for LLM Inference

GPU Specification All experiments with LLaMA3 were conducted on NVIDIA A100-SXM4 40GB GPUs, while Qwen3 experiments were run on NVIDIA L40S 48GB GPUs.

Maximum Length. The maximum context length is set to 32,768 tokens (the upper limit of Qwen3) for all LITS roles and for CoT. In practice, 2,048 tokens are sufficient for all GSM8K instances, whereas Math500 often requires longer inputs and extended reasoning chains. For the policy models in ReST and ToT-BS, we instruct the model to keep each reasoning step within 1,000 tokens.

For BN evaluation, we enforce a maximum of 1,000 output tokens for the aggregator model used in BN-SC[1]. For LLM_{eq} (BN-SC[2]) and LLM_{bn} (BN-DP), the same limit is applied, but only single-word outputs are accepted: "yes/no" for LLM_{eq}, or a number between 1–4 for LLM_{bn}. Special tokens are ignored in this check.

Temperature. We follow the settings from prior work: RAP uses temperature 0.8 for LITS roles, ReST uses 0.7, and ToT-BS reuses the ReST policy with temperature 0.7.

I Raw Efficiency and Effectiveness Results

For completeness and reproducibility, we report the raw numbers underlying the relative efficiency tables in the main paper. These include input tokens, output tokens, number of invocations, policy runtime, total runtime, and accuracy.

J Additional Failure Analyses

J.1 Instance-Level Visualization

In this section we provide the full set of instance-level visualizations for all failure cases of BN methods. Each figure reports the difference in the number of invocations between BN methods and the baseline across 100 instances, with filled markers indicating correct predictions and hollow markers incorrect ones.

J.2 CiT Regression

We analyze the correlation between reasoning tree structure and policy token overhead across different Chain-in-Tree (CiT) configurations. Token overhead is defined as the difference in policy output tokens between CiT and baseline BFS (CiT minus BFS), where positive values indicate efficiency regressions. Note that token overhead measures only policy invocations; the cost of BN evaluators is not included in this analysis.

We examine three metrics as potential predictors of token overhead:

- **max_depth**: The maximum depth of the reasoning tree, representing the longest chain of reasoning steps explored.
- **branching_factor**: The average number of nodes per depth level, representing the width of exploration at each step.
- **level**: The problem difficulty rating (1–5 for MATH500, not applicable for GSM8K), tested using Spearman rank correlation.

Table 16 summarizes the correlation between these metrics and token overhead across four experimental configurations. We use BN-SC^1 to denote the aggregator-based self-consistency method and BN-SC^2 for the pairwise self-consistency method. The superscript $^-$ indicates using an unreliable BN evaluator (Llama 8B, same as the policy model), while $^+$ indicates a reliable evaluator (Qwen 32B).

The results reveal several consistent patterns across all configurations:

Tree structure predicts overhead, problem difficulty does not. Across all four configurations, max_depth and branching_factor show significant positive correlations with token overhead ($p < 0.05$), while problem difficulty level shows no significant correlation ($\rho \approx 0.08$–0.12, $p > 0.24$ in all cases). This indicates that efficiency regressions are driven by search dynamics rather than inherent problem characteristics. A difficult problem does not necessarily lead to higher overhead if the search terminates efficiently; conversely, an easy problem can cause regression if the continuation mechanism fails to recognize early termination opportunities.

BN evaluator reliability has limited impact on correlation patterns. Comparing BN-SC^{1+} ($r = 0.294$) with BN-SC^{1-} ($r = 0.248$) on the same MATH500 dataset, we observe similar correlation strengths regardless of evaluator reliability. This suggests that the relationship between tree structure and policy token overhead is relatively stable across evaluator configurations. The slightly higher correlation with the reliable evaluator may indicate that better continuation decisions lead to more consistent (and thus more predictable) search behavior, though the difference is modest.

Dataset characteristics influence correlation magnitude. BN-SC^{1-} on GSM8K exhibits substantially stronger correlations ($r = 0.796$ for max_depth, $r = 0.681$ for branching_factor) compared to all MATH500 configurations ($r = 0.25$–0.44). This difference likely reflects GSM8K's more uniform problem structure: simpler arithmetic problems lead to more predictable search patterns, where tree size directly translates to token usage. MATH500's diverse problem types (algebra, geometry, number theory, etc.) introduce greater variance in how tree structure relates to computational cost.

Implications While tree structure metrics show promise as indicators of efficiency regression risk, their practical application for early termination requires problem-specific calibration and further investigation into the distributional characteristics of reasoning trajectories. We note several important caveats:

- The appropriate thresholds for early termination are likely to be **problem-dependent** and

Method	GSM8K Out	Inv	Time	Total	Acc	Math500 Out	Inv	Time	Total	Acc
ToT-BS	75.6k	738	1.63H	2.25H	0.98	594.5k	1266	19.6H	20.96H	0.87
+BN-SC^1	65.7k↓	651↓	1.39H↓	1.89H↓	0.96↓	423.3k↓	1230↓	12.40H↓	14.14H↓	0.89↑
+BN-SC^2	49.0k↓	465↓	1.01H↓	1.25H↓	0.96↓	221.3k↓	756↓	5.34H↓	6.03H↓	0.84↓
+BN-DP	16.4k↓	160↓	0.35H↓	0.51H↓	0.97↓	125.5k↓	253↓	4.29H↓	4.57H↓	0.86↓
ReST	83.5k	836	1.98H	1.98H	0.97↓	491.0k	1574	11.98H	11.99H	0.87=
+BN-SC^1	61.2k↓	605↓	1.30H↓	2.22H↑	0.97↓	287.9k↓	886↓	7.55H↓	8.97H↓	0.84↓
+BN-SC^2	48.7k↓	468↓	1.01H↓	1.66H↓	0.97↓	192.9k↓	661↓	4.93H↓	6.15H↓	0.85↓
+BN-DP	17.0k↓	166↓	0.36H↓	0.39H↓	0.97↓	86.4k↓	238↓	2.06H↓	2.11H↓	0.88↑
CoT	46.2k	100	0.96H	0.96H	0.96	78.1k	100	1.61H	1.61H	0.79

(a) Qwen3 32B.

Method	GSM8K Out	Inv	Time	Total	Acc	Math500 Out	Inv	Time	Total	Acc
ToT-BS	108.6k	1509	0.89H	6.91H	0.79	458.1k	2816	3.76H	14.68H	0.39
+BN-SC^1	124.8k↑	1771↑	1.01H↑	5.42H↓	0.71↓	617.0k↑	3877↑	5.14H↑	18.42H↑	0.35↓
+BN-SC^2	84.3k↓	1171↓	0.71H↑	1.32H↓	0.64↓	281.2k↓	2119↓	2.31H↓	3.44H↓	0.30↓
+BN-DP	33.9k↓	438↓	0.28H↓	1.84H↓	0.73↓	137.8k↓	1032↓	1.14H↓	4.58H↓	0.27↓
+BN-SC^1**+**	105.4k↓	1533↑	0.88H↓	7.07H↑	0.80↑	627.1k↑	3534↑	5.28H↑	23.92H↑	0.38↓
+BN-SC^2**+**	76.9k↓	1058↓	0.64H↓	1.88H↓	0.76↓	830.0k↑	3523↑	7.05H↑	13.96H↓	0.39(=)
+BN-DP+	39.1k↓	460↓	0.33H↓	2.01H↓	0.77↓	284.9k↓	1751↓	2.37H↓	8.83H↓	0.36↓
ReST	130.4k	1828	1.06H	10.44H	0.80	640.8k	3976	5.41H	25.98H	0.37
+BN-SC^1**-**	100.7k↓	1352↓	0.83H↓	6.74H↓	0.70↓	491.4k↓	3281↓	4.20H↓	20.89H↓	0.33↓
+BN-SC^2**-**	68.0k↓	960↓	0.56H↓	3.21H↓	0.73↓	238.5k↓	1944↓	1.98H↓	7.53H↓	0.29↓
+BN-DP-	29.4k↓	406↓	0.25H↓	2.63H↓	0.68↓	112.8k↓	731↓	0.95H↓	5.04H↓	0.36↓
+BN-SC^1**+**	76.9k↓	1082↓	0.63H↓	6.67H↓	0.84↑	221.1k↓	1722↓	1.81H↓	12.87H↓	0.43↑
+BN-SC^2**+**	65.0k↓	875↓	0.54H↓	3.38H↓	0.80=	386.0k↓	2053↓	3.20H↓	12.01H↓	0.34↓
+BN-DP+	27.6k↓	354↓	0.23H↓	2.47H↓	0.76↓	213.4k↓	1285↓	1.78H↓	9.10H↓	0.37=
CoT	20.0k	100	0.16H	–	0.68	121.5k	316	1.01H	–	0.34

(b) LlaMa3 8B Instruction. **BN-SC-**, **BN-DP-** uses the incompetent small LLaMA3-8B for the action aggregator or BN judge, respectively, whereas **BN-SC+** and **BN-DP+** employ the stronger Qwen3-32B for these roles.

Table 14: Efficiency and effectiveness of of ToT-BS (or ReST-MCTS*) +CiT on GSM8K and Math500. **Out** = output tokens (policy), **Inv** = number of model invocations (policy), **Time** = wall-clock running time in hours (policy), **Time (Total)** = overall LLM running time in hours (LLMs as policy, reward models, transition models and BN evaluators), **Acc** = accuracy.

dataset-dependent. The optimal depth limit for GSM8K problems may differ substantially from that for MATH500 problems due to differences in problem complexity and solution structure.

- Establishing concrete early-stopping criteria would require additional analysis of the **distribution of step lengths** across successful and unsuccessful problem instances, which is beyond the scope of the current analysis.

- The correlations reported here are **post-hoc observations** on completed search trajectories. Whether these metrics can be effectively used for **online early detection** during search

Method	GSM8K					Math500				
	Out	Inv	Time	Total	Acc	Out	Inv	Time	Total	Acc
RAP	67.8k	4606	0.69H	5.53H	0.61	70.4k	3583	0.67H	8.03H	0.18
+BN-SC+	23.5k↓	1388↓	0.24H↓	2.88H↓	0.48↓	56.7k↓	2347↓	0.61H↓	7.84H↓	0.27↑
+BN-SC^2+	14.9k↓	1128↓	0.15H↓	1.06H↓	0.46↓	25.8k↓	1449↓	0.26H↓	3.18H↓	0.21↑
+BN-DP+	7.7k↓	625↓	0.08H↓	1.25H↓	0.57↓	14.7k↓	697↓	0.14H↓	3.15H↓	0.26↑
CoT	69.4k	100	0.57H	–	0.32	71.2k	100	0.57H	–	0.17

Table 15: Efficiency and effectiveness of RAP+CiT on GSM8K and Math500.

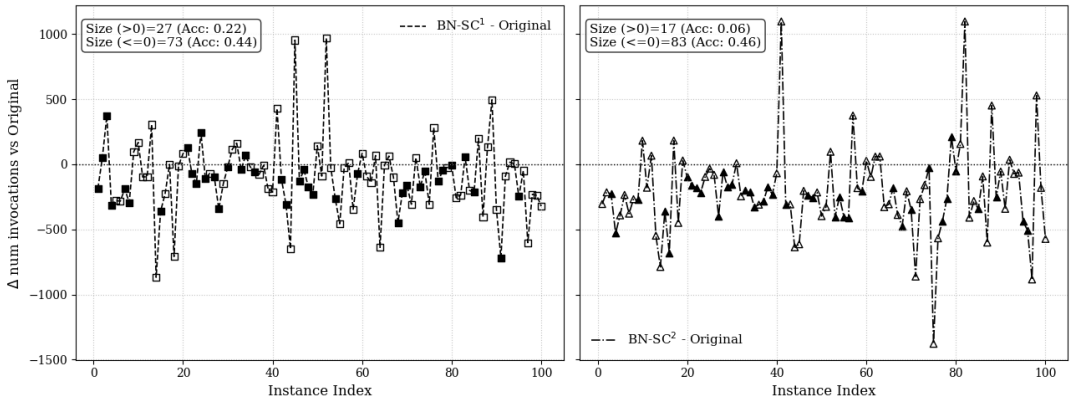


Figure 4: Instance-level analysis of a failure case on Math500 with LLaMA+Qwen (policy = LLaMA, BN = Qwen). Each point shows the relative change in the number of invocations compared to the baseline; negative values indicate higher efficiency (fewer output tokens required). Filled markers correspond to correct predictions, while hollow markers correspond to incorrect ones.

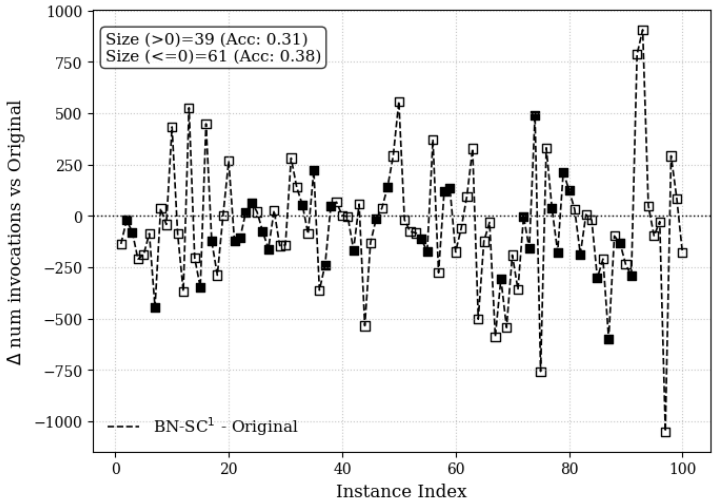


Figure 5: Instance-level analysis of failure for Math500 with LLaMA+LLaMA (BN-SC^1).

remains an open question that requires further empirical validation.

K LLM Usage

Large Language Models (LLMs) were used in this work as **writing assistants** to support clarity and style. Their role was confined to the revision stage of the manuscript and did not extend to research ideation, algorithm design, proofs, experiments, or interpretation of results.

The revision process followed a structured, iterative workflow:

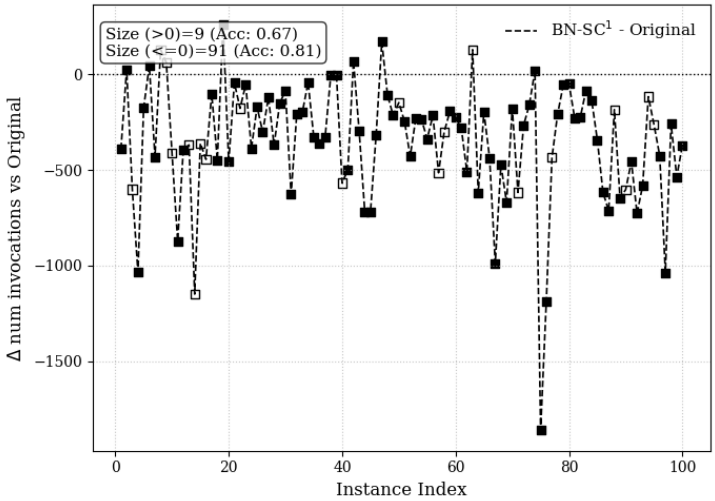


Figure 6: Instance-level analysis of failure for GSM8K with LLaMA+Qwen (BN-SC[1]).

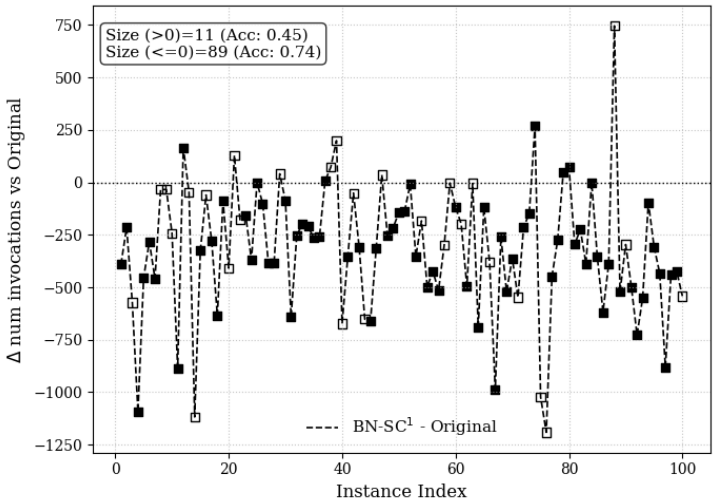


Figure 7: Instance-level analysis of failure for GSM8K with LLaMA+LLaMA (BN-SC[1]).

1. **Abstract revision:**
   - The LLM was asked to revise the abstract.
   - The authors manually revised the generated text, compared it against the original version, and selectively incorporated improvements.
   - For specific sentences, the LLM was prompted in-quote to suggest localized adjustments (e.g., rephrasing for clarity or conciseness).
   - The finalized abstract was authored by the authors after reviewing and merging both versions.
2. **Section-by-section revision:**
   - With the finalized abstract fixed, the same process was repeated sequentially for each section: Section 1, Section 2, Section 3, Section 4, Section 5, Section 6, and the Appendix.
   - At each stage: (i) the LLM was asked to revise the draft section, (ii) the authors manually revised and compared the LLM output with the original draft, and (iii) the LLM was used for fine-grained, sentence-level in-quote adjustments where needed.
   - The revised section was only finalized after careful author editing and approval.
3. **Formatting assistance:**
   - The LLM was occasionally used to generate LaTeX table skeletons, figure cap-

Configuration	Dataset	max_depth (r)	branching_factor (r)	level (ρ)
BN-SC1$^-$ (Llama 8B BN)	GSM8K	0.796***	0.681***	0.072
BN-SC1$^+$ (Qwen 32B BN)	MATH500	0.294**	0.357***	0.086
BN-SC1$^-$ (Llama 8B BN)	MATH500	0.248*	0.267**	0.084
BN-SC2$^+$ (Qwen 32B BN)	MATH500	0.436***	0.146	0.117

Table 16: Correlation between tree structure metrics and token overhead across CiT configurations. * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

tions, and consistent notation formatting. All outputs were verified and edited by the authors.

This procedure ensured that the LLM functioned only as a **linguistic and formatting assistant**. All scientific content—including the conception of the *Chain-in-Tree* framework, technical derivations, algorithmic innovations, and empirical analysis—originated entirely from the authors.

The authors take **full responsibility** for the correctness, originality, and integrity of all content.

L Package Design

The LangAgent Python package modularizes LLM-profiled roles across various search frameworks.

Role Modularity. Specifically, LangAgent organizes all LLM-mediated reasoning roles behind stable interfaces so that different search frameworks can reuse the same components without behavioral drift. Core contracts for the *world model*, *policy*, and *reward/evaluator* roles are declared once in `langagent/reasoner_base.py`, together with the shared `BaseSearchConfig` structure that wires termination and continuation settings across algorithms.

Concretely, `WorldModel` methods are defined for initialization, state transitions, and terminal checks. The policies inherit from the common `QAPolicy` skeleton, while evaluator variants specialize `QAEvaluator.fast_reward` to add correctness or usefulness signals.

Functional Search Pipelines. Search algorithms in `langagent/search` follow a functional design that keeps each phase explicit and composable. Breadth-first search exposes pure helpers for expansion, continuation-aware expansion and terminal checks that the top-level `bfs_topk` function orchestrates. Likewise, the Monte Carlo tree search implementation separates selection, expansion, simulation, and backpropagation.

Identical role objects can drive either algorithm, and new planners can reassemble the same primitives to instrument alternative search behaviors without duplicating logic.

Utilities. Auxiliary utilities—answer extraction, dataset loading, and evaluation—live in `langagent/langreason/common.py`, ensuring that any new framework automatically gains the same output handling.

M BlocksWorld Experiments

M.1 Experimental Setting

Setup of Tree Search. RAP has been employed in the BlocksWorld evaluation, while ToT-BS is not included because it was not designed for action-based planning tasks and has not been adapted to BlocksWorld-style environments in prior work.

RAP requires a scalar reward or value signal to guide tree search, but does not assume a specific implementation. While the original RAP (Hao et al., 2023) derives this signal from token-level logits, such access is unavailable in API-based LLM settings. We therefore adopt an explicit LLM-based step evaluator that directly scores state–action pairs. This evaluator serves the same functional role as a reward model and is a standard design choice in planning-oriented LLM evaluations (Zhou et al., 2024; Yao et al., 2023a) where internal model logits are inaccessible.

To encourage exploration within MCTS, we make two modifications to the default RAP settings. First, we set the self-consistency (SC) threshold to 0.99, so that the model continues chaining only when the policy consistently generates the same action; otherwise, branching is triggered. Second, we increase the LLM temperature to 1. All other hyperparameters are kept identical to those used in language reasoning tasks. Despite these changes, we observe that BN-SC still exhibits limited exploration in planning-oriented tasks such as BlocksWorld.

Use of Claude Models For the BlocksWorld experiments, we use Claude models accessed via a standard API interface due to resource availability. Since CiT is model-agnostic and operates purely at the inference level, the choice of backbone LLM does not affect the applicability of the method. Accordingly, we do not compare absolute performance across different models, and instead focus on **relative efficiency gains within the same backbone**.

Testing Data Due to API cost constraints, we evaluate all methods on a random subset of 10 BlocksWorld instances. While this sample size is insufficient for drawing strong statistical conclusions about absolute performance, it is adequate for our purpose of analyzing *relative efficiency–accuracy trade-offs* under a fixed model and environment. Since all methods are evaluated on the same instances with identical API configurations, the observed reductions in token usage and inference cost are stable and indicative of the core behavior of CiT-style search control.

M.2 Results

Table 17 reports raw token usage, inference cost, and accuracy on BlocksWorld, while Table 18 summarizes relative changes with respect to RAP.

Efficiency. BN-SC achieves substantial efficiency gains over RAP. Compared to RAP, BN-SC reduces input tokens by 79.9% and output tokens by 83.8%, leading to an 81.2% reduction in total API inference cost (from $8.86 to $1.67). This cost closely approaches that of the pure chaining baseline (CoT), indicating that BN-SC effectively collapses the search tree into a near-linear reasoning trajectory when branching is suppressed.

In contrast, BN-DP provides more moderate efficiency improvements, reducing input tokens by 19.5%, output tokens by 25.0%, and total cost by 21.4%, while still maintaining non-trivial search behavior.

Accuracy. BN-DP maintains most of the planning capability of RAP, achieving 40% accuracy compared to 50% for RAP. This suggests that explicit step-level evaluation can still guide effective search, albeit with some loss in solution quality.

BN-SC converges to the performance of the chaining baseline, achieving 20% accuracy, identical to CoT. This behavior reflects a regime where branching is largely suppressed and the search trajectory becomes effectively linear.

Takeaway. Although BlocksWorld differs substantially from mathematical reasoning, these results reinforce the same conclusions observed in the main experiments. First, CiT remains effective in reducing the computational cost of tree search by suppressing unnecessary branching. Second, the effectiveness of CiT strongly depends on the quality of BN evaluation: in this planning setting, BN-SC produces unreliable BN decisions, which suppress branching excessively and consequently reduces the effectiveness of CiT. Importantly, these observations are consistent with the findings reported in the main text.

	BlocksWorld			
Method	In	Out	Cost	Acc
ReAct	151,709	6,092	$0.5465	0.20
RAP	1,960,749	198,543	$8.8604	0.50
+BN-DP	1,578,131	148,834	$6.9669	0.40
+BN-SC	393,993	32,252	$1.6658	0.20

Table 17: Raw input/output token usage, API inference cost, and accuracy on BlocksWorld using Claude 3.5 Sonnet. These values are used to compute the relative changes in Table 18.

	BlocksWorld			
Method	In	Out	Cost	Acc
Relative to RAP	(baseline Acc: **0.50**; ReAct: 0.20)			
BN-DP	19.5%↓	25.0%↓	21.4%↓	20.0%↓
BN-SC	79.9%↓	83.8%↓	81.2%↓	60.0%↓

Table 18: Relative changes in input tokens, output tokens, inference cost, and accuracy compared to RAP on BlocksWorld using Claude 3.5 Sonnet. ↓ denotes decrease.

N Full Math500 Evaluation (316 Instances)

To verify that the efficiency–accuracy trends reported in the main experiments (based on the first 100 instances) are not driven by subset variance, we additionally evaluate on all 316 filtered Math500 instances using Qwen-32B under both ReST-MCTS and ToT-BS settings.

Table 19 reports the results. Under ReST-MCTS, BN-DP reduces total runtime from 49.43H to 9.16H (≈5.4× reduction) while accuracy changes from 0.86 to 0.88. Under ToT-BS, runtime decreases from 42.42H to 8.69H (≈4.9× reduction)

while accuracy changes from 0.896 to 0.87. These results confirm that the efficiency gains and accuracy preservation observed on the 100-instance subset generalize to the full filtered benchmark.

Method	Total Runtime	Accuracy
ReST-MCTS (Qwen-32B)		
Baseline	49.43H	0.86
+BN-DP	9.16H	0.88
ToT-BS (Qwen-32B)		
Baseline	42.42H	0.896
+BN-DP	8.69H	0.87

Table 19: Efficiency and accuracy on all 316 filtered Math500 instances with Qwen-32B. BN-DP achieves 4.9–5.4$\times$ runtime reduction with negligible accuracy change, consistent with the 100-instance results in the main paper.