Chopping Trees: Semantic Similarity Based Dynamic Pruning for Tree-of-Thought Reasoning

Joongho Kim[1,*] **Xirui Huang**[1] **Zarreen Reza**[1,†] **Gabriel Grand**[2,†]
Kevin Zhu[1,†] **Ryan Lagasse**[1,†]
[1]Algoverse AI Research [2]Massachusetts Institute of Technology (MIT)
joonghokim@google.com, kevin@algoverse.us

Abstract

Tree-of-Thought (ToT) reasoning boosts the problem-solving abilities of Large Language Models (LLMs) but is computationally expensive due to **semantic redundancy**, where distinct branches explore equivalent reasoning paths. We introduce **Semantic Similarity-Based Dynamic Pruning (SSDP)**, a lightweight method that, to the best of our knowledge, is the first framework to integrate online semantic merging into parallelized tree search, enabling the clustering and pruning of redundant steps in real time. Across reasoning benchmarks, including GSM8K and MATH500, SSDP achieves up to a **2.3x speedup** over state-of-the-art tree-search baselines while maintaining competitive accuracy (typically within 5% of the strongest baseline) and reducing the number of explored nodes by **85–90%**, demonstrating a practical approach to efficient, scalable LLM reasoning. The implementation of SSDP is publicly available at `https://github.com/kimjoonghokim/SSDP`.

1 Introduction

Tree-of-Thought (ToT) search strategies [Yao et al., 2023] enable Large Language Models to explore multiple reasoning paths beyond linear Chain-of-Thought prompting [Wei et al., 2022], improving performance on complex problems requiring backtracking or strategic lookahead. However, exploring vast reasoning trees remains computationally expensive, with many branches redundantly exploring semantically equivalent reasoning paths.

We introduce **Semantic Similarity-Based Dynamic Pruning (SSDP)**, a lightweight method that identifies and merges semantically similar reasoning branches into a single representative "hypernode" during search. SSDP requires only a reward model and its core semantic merging logic to dynamically prune the search space, without dataset-specific fine-tuning. To the best of our knowledge, SSDP is the first framework that integrates online semantic merging, that is, merging semantically similar reasoning paths dynamically during inference rather than after completion, into parallelized tree search to cluster and prune redundant steps in real time.

Across challenging benchmarks like GSM8K and MATH500, SSDP achieves up to a **2.3x speedup** over strong tree-search baselines while maintaining comparable accuracy. It also dramatically reduces the search space, exploring on average **85-90% fewer nodes**, demonstrating that targeting semantic redundancy is an effective and practical strategy for scalable LLM reasoning.

Our contributions are threefold:

[*]Lead Author.
[†]Senior Authors.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Efficient Reasoning.

1. We propose SSDP, a novel and lightweight pruning method that leverages semantic similarity to make tree-based search more efficient and, to the best of our knowledge, is the first framework of its kind used for inference time scaling.
2. We provide extensive empirical evidence showing that SSDP substantially reduces inference time and computational load across multiple models and benchmarks, without a significant trade-off in accuracy.
3. We demonstrate that explicitly targeting semantic redundancy is a highly effective strategy for optimizing LLM reasoning at inference time, paving the way for more scalable and practical applications of advanced reasoning frameworks.

2 Related Work

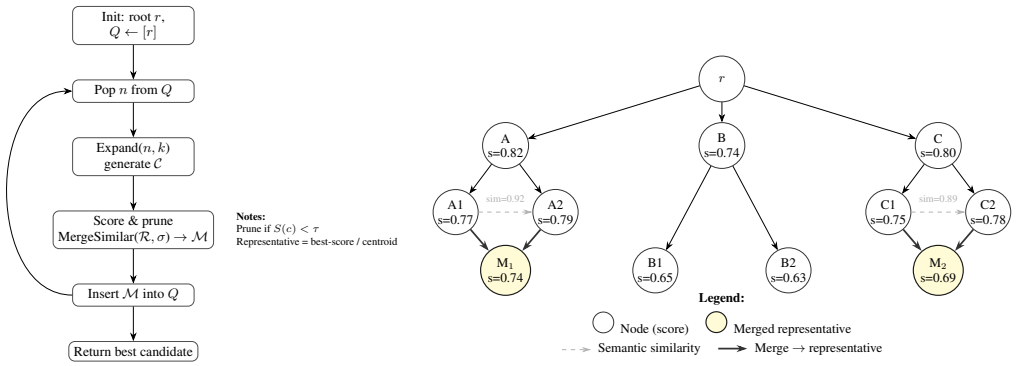
Our work lies at the intersection of tree search for LLM reasoning, efficiency-focused pruning and merging, and semantic similarity for reasoning.

Tree Search for LLM Reasoning. Linear strategies such as Chain-of-Thought (CoT) [Wei et al., 2022] and Self-Consistent CoT [Wang et al., 2023] improved multi-step reasoning but explored limited paths. Tree-of-Thoughts (ToT) [Yao et al., 2023] expanded exploration using sampling, beam search, and MCTS [Cobbe et al., 2021, Hao et al., 2023, Wan et al., 2024], aligning with the "inference scaling law" [Hong et al., 2024, Lin et al., 2024, Fu et al., 2024, Cai et al., 2024]. Dynamic Parallel Tree Search (DPTS) [Ding et al., 2025] improved parallel efficiency but still explored many redundant states. **SSDP addresses this by pruning redundant paths before they are generated.** Recent work has acknowledged that tree-based methods continue to suffer from computational overheads [Rufail et al., 2025], reinforcing the need for efficiency-focused optimizations like SSDP.

Pruning and Merging for Efficiency. Pruning improves reasoning even in linear CoT [Zhao et al., 2025]. Methods like FETCH [Wang et al., 2025] merge semantically similar states but require fine-tuned models. **SSDP provides a lightweight, general alternative**, using only a small reward model and semantic merging to achieve large efficiency gains without task-specific tuning.

Semantic Similarity in Verification. Semantic Self-Consistency [Knappe et al., 2025] clusters completed reasoning chains post-hoc. In contrast, **SSDP integrates semantic merging during tree growth**, shaping the search process itself to make reasoning faster and more efficient.

3 Methodology



(a) **Figure 1a:** SSDP flowchart.

(b) **Figure 1b:** SSDP tree.

Figure 1: **Overview of SSDP.** (**a**) Flowchart of the SSDP algorithm showing its iterative expand–score–prune–insert loop. (**b**) Example search tree where semantically similar nodes are merged into representative clusters, reducing redundant exploration.

In this section, we present the design of **Semantic Similarity–based Dynamic Pruning (SSDP)**. We first describe the pruning-augmented tree search framework, then introduce the semantic similarity merging mechanism, detail the node-level scoring and pruning strategy, and finally present our search and stopping criteria.

An overview of the SSDP process is shown in Figure 1a (flowchart) and Figure 1b (tree example). Figure 1a illustrates the iterative tree expansion and pruning loop, while Figure 1b visualizes how semantically similar nodes are merged into representative clusters during search.

3.1 Framework Overview

SSDP is designed to improve tree-based LLM reasoning efficiency by pruning semantically redundant nodes. It builds upon the parallel tree search framework of DPTS [Ding et al., 2025], inheriting dynamic expansion control.

The search loop follows standard MCTS-style stages: selection of promising nodes, expansion to generate children, evaluation using a reward model, semantic merging and pruning (our main contribution), and backpropagation of node scores. This process reduces redundant reasoning paths and leverages parallelism for efficient computation.

The search loop proceeds as follows:

1. **Selection:** The framework first selects up to k promising leaf nodes from the search frontier to expand in parallel. This selection is guided by an Upper Confidence Bound (UCB) policy, defined by [Ding et al., 2025] as:
$$\mathrm{UCB}(n) = \frac{Q_n}{N_n} + w\sqrt{1 + N_{\mathrm{parent}(n)}} \cdot \frac{\phi(n)}{1 + N_n}$$
where Q_n is the cumulative reward, N_n is the visit count, $\phi(n)$ is a node's intrinsic score from a reward model, and w is an exploration weight.
2. **Expansion and Evaluation:** For each selected node, the base LLM generates b candidate children via temperature sampling. Each new child node is then evaluated by a reward model to obtain its score.
3. **Semantic Merging:** Before the newly generated children are added to the search frontier, they undergo our semantic merging process. As detailed in Section 3.2, this step identifies and consolidates semantically equivalent nodes into a single representative. This fundamentally reduces the effective branching factor of the search tree, preventing the system from wasting resources on redundant reasoning paths.
4. **Backpropagation:** Finally, the reward from the newly evaluated nodes is propagated up the tree to update the statistics (Q_u, N_u) of their ancestors, following the standard MCTS procedure.

This process is further accelerated by parallel execution and generic efficiency mechanisms, such as early stopping for unpromising rollouts.

3.2 Semantic Merging Module

The core innovation of SSDP is its online semantic merging module, which prunes the search tree at each expansion step. This process consists of three stages: embedding, clustering, and representative selection.

1. **Embedding Generation:** Each child node c is encoded into a dense vector $s(c) = \mathcal{E}(\text{text}(c))$ using a frozen sentence-transformer. The resulting embeddings are ℓ_2-normalized to facilitate efficient cosine similarity calculations.

2. **Similarity-Based Clustering:** Sibling nodes with cosine similarity above threshold τ are grouped into clusters of semantically equivalent nodes.

3. **Representative Selection:** From each cluster C, only the node with highest reward score $\phi(c)$ is retained; all others are pruned to free memory. The unique representatives $\{c_1^*, c_2^*, \dots\}$ are added to the frontier.
$$c^* = \arg\max_{c \in C} \phi(c)$$

All other nodes within the cluster are discarded, and their associated KV caches are immediately freed to reduce memory overhead. Only the set of unique representatives $\{c_1^*, c_2^*, \ldots\}$ is added to the search frontier. This aggressive, online pruning strategy is the primary mechanism through which SSDP reduces the number of nodes explored and accelerates the time to solution.

4 Experimental Setup

Our SSDP implementation builds on the official Dynamic Parallel Tree Search (DPTS) codebase [Ding et al., 2025], adding semantic merging during node expansion while retaining other architectural components for fair comparison.

4.1 Models, Datasets, and Baselines

Models. We evaluate four open-source, instruction-tuned models from the Llama and Qwen families: `Llama-3.1-8B-Instruct` [Team, 2024], `Llama-3.2-3B-Instruct`, `Qwen2.5-1.5B-Instruct`, and `Qwen2.5-7B-Instruct` [Bai et al., 2024]. These were chosen to ensure our findings are robust across different model architectures and scales.

Datasets. Experiments were conducted on two standard multi-step reasoning benchmarks: **GSM8K** [Cobbe et al., 2021] and **MATH500** [Hendrycks et al., 2021]. These datasets have canonical scoring procedures, enabling direct comparison with prior work.

Baselines. We compare SSDP against four widely-used search methods to quantify its impact: (1) **DPTS** [Ding et al., 2025], the state-of-the-art parallel search framework; (2) **MCTS**, a standard Monte Carlo Tree Search implementation for LLM reasoning [Yao et al., 2023]; (3) **Best-of-N**, which samples independent CoT traces and selects the best result [Cobbe et al., 2021]; and (4) **Beam Search**, which maintains a fixed-size beam of the most promising reasoning paths [Yao et al., 2023].

4.2 Evaluation Protocol

For a fair comparison, all methods were run with identical prompts, reward models, and matched computational budgets (e.g., maximum expansions). Our primary evaluation metrics are:

- **Accuracy:** Canonical exact-match scoring for each dataset.
- **Time (s):** Average inference time required to process a sample in the dataset.

As a secondary analysis, we also measured the average number of nodes generated and explored per sample for DPTS and SSDP using Qwen-2.5 1.5B. This secondary experiment on Qwen-2.5 1.5B examines node exploration efficiency to help explain observed performance differences (Table 3).

5 Results

Across the evaluated methods and benchmarks, SSDP consistently reduced inference latency while preserving the quality of the final answer. Table 1 reports the accuracy and inference times of the models for each method. In terms of correctness, SSDP attains parity with the strongest baselines: across GSM8K the SSDP accuracies closely track DPTS, Beam and Best-of-N, and on MATH500 SSDP matches or outperforms several non-DPTS baselines in multiple model settings. In summary, we do not observe systematic degradation in the accuracy of the final answer attributed to semantic merging.

The runtime improvements from SSDP are substantial and consistent. Aggregating across the four methods and two data sets (Table 2), SSDP reduces the total inference time by approximately $2.3\times$, $5.2\times$, $5.9\times$, and $7.7\times$, when compared to DPTS, Best-of-N, Beam and MCTS respectively. These ratios indicate that SSDP eliminates a large fraction of redundant model computation, yielding a significantly faster completion for every search strategy tested.

We also observe modest model-family differences: the proportional speedups are slightly larger for the Llama family than for Qwen in our runs ($T_{\text{baseline}}/T_{\text{SSDP}} \approx 5.87$ for Llama vs. ≈ 4.77 for Qwen; Table 2). The values are calculated by taking the mean SSDP speedup ratio against all four search methods and across both datasets for models belonging to the same family. This pattern is consistent

Table 1: Comparisons across search algorithms on LLM reasoning tasks. (Higher score is better).

Model	Method	MATH500		GSM8K	
		Acc. (%)	Time (s)	Acc. (%)	Time (s)
Qwen-2.5 1.5B	MCTS†	56.6	117.37	75.1	73.28
	Best-of-N†	52.6	89.87	70.1	33.37
	Beam†	52.4	104.58	71.5	41.27
	DPTS	58.6	37.4	80.9	14.7
	SSDP	58.24	**19.01**	75.54	**6.43**
Qwen-2.5 7B	MCTS†	75.2	121.46	89.6	79.68
	Best-of-N†	71.6	91.29	88.2	34.89
	Beam†	72.4	106.89	86.7	36.49
	DPTS	76.3	44.5	89.5	19.9
	SSDP	75.56	**22.35**	87.73	**7.08**
Llama-3 3B	MCTS†	48.6	111.80	64.0	57.19
	Best-of-N†	46.4	91.34	57.1	27.27
	Beam†	45.2	104.36	58.4	28.27
	DPTS	56.8	39.1	57.8	9.3
	SSDP	52.47	**15.54**	56.52	**4.3**
Llama-3 8B	MCTS†	54.2	143.36	69.5	69.74
	Best-of-N†	49.8	122.63	67.6	33.48
	Beam†	49.6	142.21	68.3	34.51
	DPTS	61.9	49.8	62.4	12.8
	SSDP	60.19	**20.37**	61.83	**5.67**

† Results were obtained from the DPTS paper [Ding et al., 2025].

Table 2: Aggregate SSDP speedups

Baseline	**Implied speedup (×)**
DPTS	2.26
Best-of-N	5.20
Beam	5.94
MCTS	7.68
Model-family averages	
Llama family	5.87
Qwen family	4.77

with the intuition that models with higher per-token or per-step generation cost benefit more from avoiding redundant rollouts, although the magnitude depends on decoding and batching settings.

Table 3: Average nodes generated and explored per sample (Qwen2.5-1.5B)

Method	**MATH500**		**GSM8K**†	
	Nodes gen.	Nodes expl.	Nodes gen.	Nodes expl.
DPTS	214.4	53.3	79.1	19.5
SSDP	**31.1**	**11.5**	**9.4**	**4.8**

† Experiment was performed on a 500 sample subset of GSM8K.

Diagnostic counts corroborate the mechanistic source of these gains. Table 3 shows that, for Qwen-2.5-1.5B, SSDP generates far fewer candidates on average (MATH500: 31.1 vs. 214.4; GSM8K: 9.4 vs. 79.1) and explores substantially fewer nodes per sample (MATH500: 11.5 vs. 53.3; GSM8K: 4.8 vs. 19.5). The reduction in generated and explored nodes aligns with the observed inference time savings and supports the interpretation that sibling-level semantic merging prevents repeated model invocations on near-duplicate continuations.

Taken together, these results demonstrate that SSDP is an effective, low-cost augmentation to Tree-of-Thought reasoning processes: it achieves large, robust latency reductions by pruning semantic redundancy while maintaining final-answer accuracy.

Based on this analysis, we selected $\tau = 0.75$ as the default threshold for our experiments, as it provides a strong balance between efficiency and accuracy. However, the threshold can be adjusted depending on whether the user prioritizes speed (lower τ) or accuracy (higher τ).

5.1 Ablation Study

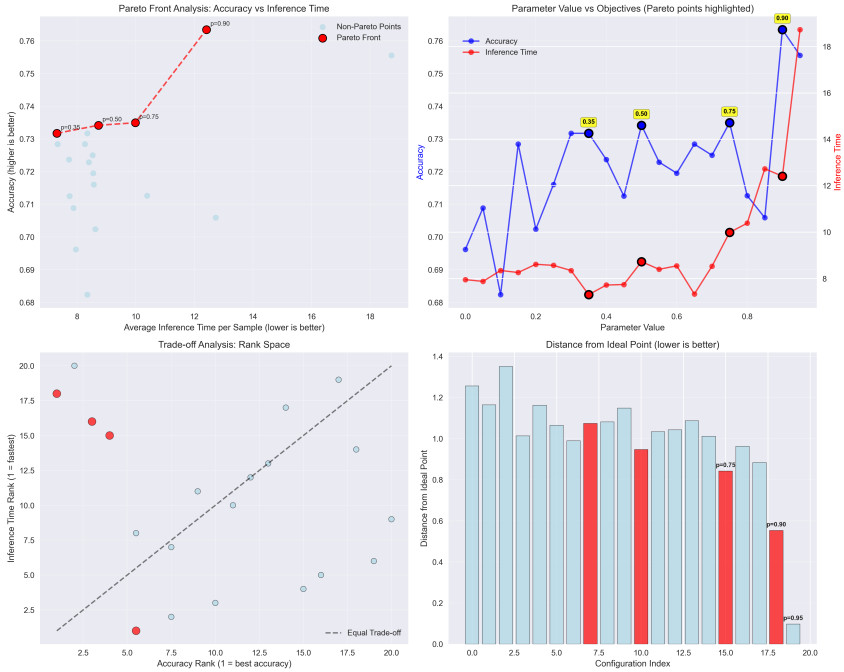


Figure 2: Pareto Front Analysis on Similarity Threshold τ

We conducted an ablation study to analyze the impact of the similarity threshold τ on the performance of SSDP. The experiments were performed on the GSM8K dataset using Qwen-2.5 1.5B, varying τ from 0 to 1 in steps of 0.05. For each threshold value, we measured both accuracy and inference time to understand the trade-off between speed and performance.

The results of this analysis reveal four threshold values that lie on the Pareto front, representing optimal trade-offs between speed and accuracy: 0.35, 0.5, 0.75, and 0.9. Specifically:

- $\tau = 0.35$: fastest configuration while still achieving some accuracy gains.
- $\tau = 0.5$ and 0.75: balanced configurations offering good trade-offs between speed and accuracy.
- $\tau = 0.9$: highest accuracy but slower inference.

6 Limitations

Semantic similarity does not imply equivalence (and vice versa). SSDP assumes that high embedding similarity between intermediate steps indicates functional redundancy, yet semantically similar text can diverge logically, and conversely, two phrasing-distant steps can be logically equivalent. This mismatch risks (i) pruning a step that later enables a distinct, correct derivation, or (ii) retaining a near-duplicate that appears dissimilar in embedding space. Practical mitigations include conservative thresholds, verifier-aware gating, and deferred merging for uncertain clusters.

Evidence concentrated on math; broader validation needed. Empirical results are centered on math-style reasoning (e.g., GSM8K, MATH500). While these benchmarks stress multi-step deduction, they may not capture the linguistic variability, world knowledge, or planning demands of other domains (code generation, multi-hop QA, commonsense, tool use). Generalization to such settings and across model sizes/families remains an open question requiring domain-specific studies and possibly task-adapted similarity encoders.

Similarity threshold (τ) is hand-tuned. The current system relies on a single, manually selected τ to decide when steps are "similar enough" to merge. This choice is dataset- and model-dependent, and small changes can shift pruning behavior noticeably. As a result, reported results may partially reflect the chosen threshold rather than an intrinsic property of the method; understanding performance across a range of τ remains important.

Potential over-pruning of distinct reasoning. When two steps read alike, the procedure may collapse them even if they pursue materially different ideas. Such removals can narrow exploration and hide alternative derivations that would have succeeded later in the search. The extent of this effect is not fully quantified here, so accuracy losses attributable to over-pruning may be underreported.

Embedding backbone cost and sensitivity. The approach inherits the computational footprint and biases of the embedding model used to judge similarity. Encoder latency and memory add overhead to search time, and domain mismatches can distort which steps appear "close." Performance and conclusions may therefore vary with the encoder choice, dimensionality, and preprocessing, which are not exhaustively analyzed in this work.

7 Conclusion

We introduced Semantic Similarity Based Dynamic Pruning (SSDP), a lightweight method to address the critical inefficiency in Tree-of-Thought (ToT) reasoning caused by semantic redundancy. By integrating an online semantic merging module into a parallel search framework, SSDP identifies and prunes redundant reasoning paths in real-time without requiring complex fine-tuning. Our experiments across multiple models and benchmarks demonstrate that this approach yields substantial gains: SSDP achieves up to a **2.3x speedup** over the parallel-search DPTS baseline and even greater gains (5-8x) over other standard search methods while maintaining comparable accuracy. This efficiency is a direct result of reducing the search space by an average of **85-90%**. By showing that a simple, general-purpose pruning mechanism can yield substantial performance gains, SSDP offers a practical path toward making inference-time scaling more efficient. It paves the way for more complex and deliberative AI reasoning to become not only more powerful but also scalable and practical for real-world applications.

References

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, 2023.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2022.

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

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Henryk Michalewski, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model. *arXiv preprint arXiv:2305.14992*, 2023.

Yecheng Wan, Geng T. M. K. K., Yifan Zhao, Chunyi Li, Hongyin Liu, Fuzhen Zhuang, and Qing He. Mcts-rollout: A monte carlo tree search-based rollout policy for real-time alphazero, 2024.

Ke Hong, Guohao Dai, Jiaming Xu, Qiuli Mao, Xiuhong Li, Jun Liu, Kangdi Chen, Yuhan Dong, and Yu Wang. Flashdecoding++: Faster large language model inference with asynchronization, flat gemm optimization, and heuristics. In P. Gibbons, G. Pekhimenko, and C. De Sa, editors, *Proceedings of Machine Learning and Systems*, volume 6, pages 148–161, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/file/5321b1dabcd2be188d796c21b733e8c7-Paper-Conference.pdf.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for llm compression and acceleration, 2024. URL https://arxiv.org/abs/2306.00978.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of llm inference using lookahead decoding, 2024. URL https://arxiv.org/abs/2402.02057.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads, 2024. URL https://arxiv.org/abs/2401.10774.

Yifu Ding, Wentao Jiang, Shunyu Liu, Yongcheng Jing, Jinyang Guo, Yingjie Wang, Jing Zhang, Zengmao Wang, Ziwei Liu, Bo Du, Xianglong Liu, and Dacheng Tao. Dynamic parallel tree search for efficient llm reasoning, 2025. URL https://arxiv.org/abs/2502.16235.

Andrew Rufail, Daniel Kim, Sean O'Brien, and Kevin Zhu. Clear: Contrasting textual feedback with experts and amateurs for reasoning, 2025. URL https://arxiv.org/abs/2504.07116.

Shangziqi Zhao, Jiahao Yuan, Guisong Yang, and Usman Naseem. Can pruning improve reasoning? revisiting long-cot compression with capability in mind for better reasoning, 2025. URL https://arxiv.org/abs/2505.14582.

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, 2025. URL https://arxiv.org/abs/2502.11183.

Tim Knappe, Ryan Li, Ayush Chauhan, Kaylee Chhua, Kevin Zhu, and Sean O'Brien. Semantic self-consistency: Enhancing language model reasoning via semantic weighting, 2025. URL https://arxiv.org/abs/2410.07839.

Llama 3.1 Team. The llama 3.1 family of models, 2024.

Jinze Bai, Shuaiqiang Wang, Zeyu Chen, Yunfei Chu, Haodong Duan, Chenglong Huang, Jianxin Ma, Hongning Wang, Hang Yan, Javis Z. Wu, Yubo Zhang, Chang Zhou, and Jingren Zhou. Qwen2: The new generation of large language models, 2024.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

A Technical Appendices and Supplementary Material

A.1 Reproducibility Statement

- **Code Availability:** All code used for inference and evaluation will be available at `https://github.com/kimjoonghokim/SSDP`.
- **Models:** The Llama 3 family of models has restricted access, but is made available by Meta on request. You can access them by requesting permission through the provided Meta license. The Qwen family of models are publicly accessible and can be found on HuggingFace
- **Datasets:** Both MATH500 and GSM8K datasets used are publicly available through HuggingFace and are also accessible through our codebase.
- **Prompts:** All prompts were taken from the DPTS paper [Ding et al., 2025].

A.2 GPU Usage

Table 4: GPUs used for code testing and running experiments.

GPU Model	Memory (GB)	Usage Purpose	Approx. Hours Used
NVIDIA H200 SXM	141	Running experiments	40
NVIDIA A40	48	Code testing and development	120

A.3 Integration with DPTS Search Loop

SSDP clustering is applied immediately after node expansion and PRM scoring:

while $t < T_{\max}$ and stopping criteria not met **do**
  $\mathcal{S} \leftarrow \textsc{Select}(\mathcal{N}_{\text{all}}, k)$ ▷ UCB-based selection
  for each node $n \in \mathcal{S}$ **do**
    $\mathcal{C}_n \leftarrow \textsc{Expand}(n, b)$ ▷ Generate b children
    for each child $c \in \mathcal{C}_n$ **do**
      $\phi(c) \leftarrow \text{PRM}(c)$ ▷ Score with reward model
      $s(c) \leftarrow \mathcal{E}(\text{decode}(x_c))$ ▷ Compute embedding
    end for
    $\mathcal{C}'_n \leftarrow \textsc{ClusterAndPrune}(\mathcal{C}_n, \tau)$ ▷ SSDP merging
    $n.\text{children} \leftarrow \mathcal{C}'_n$
  end for
  $\mathcal{N}_{\text{all}} \leftarrow \mathcal{N}_{\text{all}} \cup \bigcup_{n \in \mathcal{S}} \mathcal{C}'_n$
  $\textsc{Backpropagate}(\mathcal{S})$
end while

A.4 Early Stopping and Pruning Mechanisms

SSDP inherits DPTS's adaptive stopping criteria to prevent over-exploration:

Early Stopping: During rollout, if a node's reward $\phi(n)$ falls below a threshold $\theta_{\text{es}} = \lambda_{\text{es}} \cdot \bar{\phi}_{\text{explore}}$, where $\bar{\phi}_{\text{explore}}$ is the mean reward of explored nodes and $\lambda_{\text{es}} = 0.8$, the rollout terminates early.

Deep Seek Threshold: For exploration nodes, a stricter threshold $\theta_{\text{ds}} = \lambda_{\text{ds}} \cdot \bar{\phi}_{\text{explore}}$ (with $\lambda_{\text{ds}} = 0.8$) is applied to prune low-quality branches.

Solution Quality Gating: Once $t^* = 5$ solutions are found, new rollouts are only pursued if the selected node's reward exceeds the best known solution reward: $\phi(n) > \max_{n \in \mathcal{N}_{\text{term}}} \phi(n)$.

Time-Based Stopping: The search terminates when wall-clock time exceeds $T_{\max}$ (default: 120 seconds) or after a maximum number of rollouts (default: 20).

A.5 Hyperparameters

Table 5: SSDP Hyperparameters

Parameter	Symbol	Default Value
Similarity threshold	τ	0.75
Tree width (beam size)	b	4
Exploration weight	w	$1/\sqrt{2}$
Early stopping factor	λ_{es}	0.8
Deep seek factor	λ_{ds}	0.8
Solution count threshold	t^*	5
Max search time	$T_{\max}$	120s
Max rollouts	$R_{\max}$	20
Exploit ratio	p	0.5

A.6 Embedding Models

We evaluate the following sentence transformer architecture:

- `all-MiniLM-L6-v2`: 22M parameters, 384-dim embeddings, fast inference

The model is frozen during inference and kept on GPU for low-latency encoding.

A.7 Merging Algorithm

Algorithm 1 MERGECLUSTER(C)

Require: cluster $C = \{c_1, \ldots, c_m\}$

1: **Option A:** choose representative $r \leftarrow \arg\max_{c \in C} S(c)$

2: **Option B:** compute centroid embedding

$$\bar{e} \leftarrow \frac{1}{|C|} \sum_{c \in C} e(c)$$

and create merged node from $\bar{e}$

3: Recompute $S(r)$ (e.g., rescore or use representative's score)

4: **return** r

A.8 Reward model and prompts

All methods use the same system prompt template (instructing stepwise chain-of-thought output and final-answer extraction). Candidate traces are rescored using the Math-Shepherd reward model `peiyi9979/math-shepherd-mistral-7b-prm` for reranking and final answer selection. Rescoring with an external reward model ensures consistent selection criteria across SSDP, DPTS, Best-of-N, Beam Search, and MCTS.

A.9 SSDP Method Example

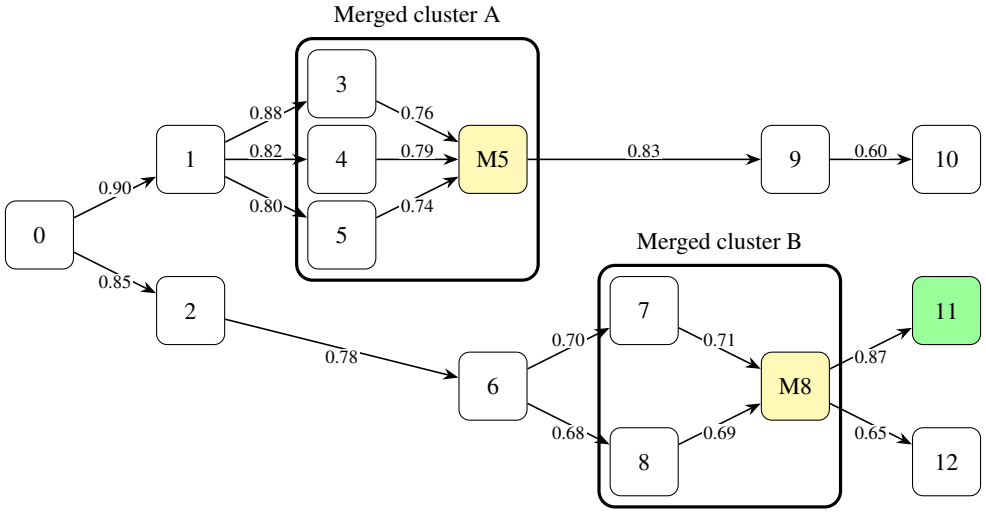


Table 6: Node numbers and short description of each reasoning step shown in the SSDP figure. Hypothetical reward model scores are attached to the arrows between nodes to simulate an end-to-end SSDP process.

Node #	Reasoning step
0	Problem statement: Find positive integer pairs (x, y) satisfying $\frac{1}{x} + \frac{1}{y} = \frac{1}{6}$ and whose sum is the smallest possible odd number.
1	Branch S1 — Algebraic rewrite: Rearrange to expose factorization structure, e.g. $(x-6)(y-6) = 36$.
2	Branch S2 — Divisor/substitution path: Express y in terms of x via $y = \frac{6x}{x-6}$ and enumerate divisors.
3	S1 child (factor pair 1): Factor $36 = 1 \times 36 \Rightarrow$ candidate $(7, 42)$ and $(9, 18)$ and $(8, 24)$.
4	S1 child (factor pair 2): Factor $36 = 2 \times 18 \Rightarrow$ candidate $(8, 24)$ and $(7, 42)$ and $(9, 18)$.
5	S1 child (factor pair 3): Factor $36 = 3 \times 12 \Rightarrow$ candidate $(9, 18)$ and $(8, 24)$ and $(7, 42)$.
M5	S1 child (factor pair 3): Factor $36 = 3 \times 12 \Rightarrow$ candidate $(9, 18)$ and $(8, 24)$ and $(7, 42)$.
6	Performing isolation for y to then extract (x,y) values.
7	S2 child (divisor trial 1): Divisor enumeration yields candidate $(12, 12)$ and $(15, 15)$.
8	S2 child (divisor trial 2): Divisor enumeration yields candidate $(10, 15)$ and $(12, 12)$.
M8	S2 child (divisor trial 2): Divisor enumeration yields candidate $(10, 15)$ and $(12, 12)$.
9	Find odd sums: Add them both together, then modulo 2 to see if =1.
10	**Incorrect Answer**: (9,18) and (7,42). This is incorrect because we are asking for the smallest sum, and both are incorrect - there is a smaller combination.
11	**Answer found:** (10,15) because only one of them is odd, and no numbers between 10 and 15 were found that can be used.
12	Reasoning still in process: Calculating the sums of 10+15 and 12+12.