Atom of Thoughts for Markov LLM Test-Time Scaling

Fengwei Teng[1,2], **Quan Shi**[3], **Zhaoyang Yu**[2], **Jiayi Zhang**[1,2],
Yuyu Luo[1], **Chenglin Wu**[2†], **Zhijiang Guo**[1†]
[1]HKUST(GZ), [2]DeepWisdom, [3]Renmin University of China

Abstract

Large Language Models (LLMs) have achieved significant performance gains through test-time scaling methods. However, existing approaches often incur redundant computations due to the accumulation of historical dependency information during inference. To address this challenge, we leverage the memoryless property of Markov processes to minimize reliance on historical context and propose a Markovian reasoning process. This foundational Markov chain structure enables seamless integration with various test-time scaling methods, thereby improving their scaling efficiency. By further scaling up the Markovian reasoning chain through integration with techniques such as tree search and reflective refinement, we uncover an emergent atomic reasoning structure, where reasoning trajectories are decomposed into a series of self-contained, low-complexity atomic units. We name this design Atom of Thoughts (AoT). Extensive experiments demonstrate that AoT consistently outperforms existing baselines as computational budgets increase. Importantly, AoT integrates seamlessly with existing reasoning frameworks and different LLMs (both reasoning and non-reasoning), facilitating scalable, high-performance inference.We submit our code alongside this paper and will make it publicly available to facilitate reproducibility and future research.

1 Introduction

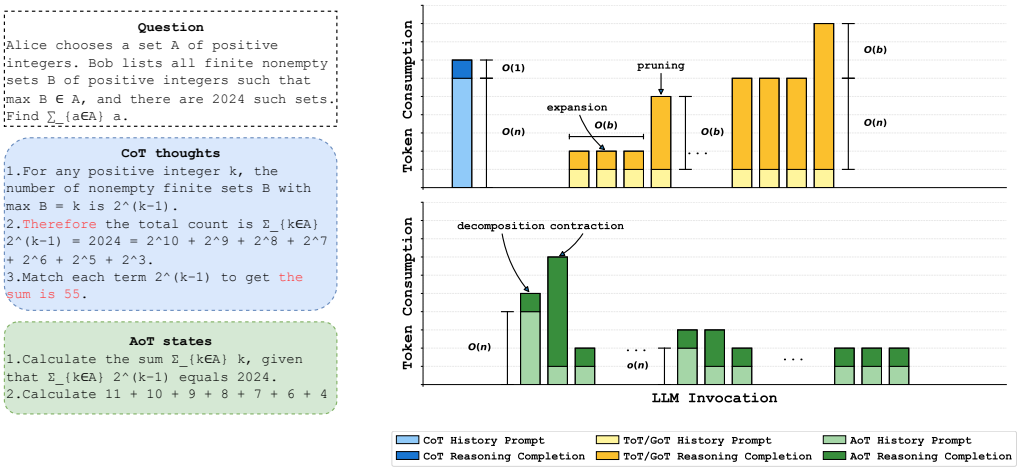
Large Language Models (LLMs) exhibit remarkable scaling behavior: as model parameters and training data increase, their performance improves predictably across a wide range of tasks [21]. Recently, test-time scaling methods have emerged to push the performance boundary further by increasing computational resources during inference. These range from basic Chain-of-Thought (CoT) prompting that extends reasoning chains [33], to more structured approaches like Tree-of-Thought (ToT) [40] and Graph-of-Thought (GoT) [4] that organize multiple LLM invocations for exploring solution spaces, and recent reasoning models such as OpenAI O1 [26] and DeepSeek R1 [9] that enhance LLMs' long-chain reasoning ability through post-training [29, 24, 18].

However, current framework-based test-time scaling methods typically rely heavily on retaining extensive historical information. Even the simplest CoT must preserve the entire reasoning trajectory to generate each subsequent step [33, 53]. Tree-based methods maintain ancestor and sibling relations for branching decisions [40, 56, 11], while Graph-based methods introduce even more complex dependencies through arbitrary node connections [4, 52]. Figure 1b analyzes these representative structures and abstract the complexity of historical information and reasoning completion token involved at each LLM invocation.

To decouple the current problem's reasoning from processing historical information and thus minimize their mutual interference during test-time computation, we aim to generalize Markov chain–style structures to general-purpose reasoning. By exploiting the **memoryless property** of Markov processes, we design the Markovian reasoning process, where each state encapsulates a self-contained

[†]Corresponding Authors. Contact: steamedbun2002@outlook.com

39th Conference on Neural Information Processing Systems (NeurIPS 2025).



(a) (b)

Figure 1: **Token Allocation Comparison in Reasoning Frameworks.** *Figure (a)* demonstrates the differences between thoughts and states, where the red-highlighted text in thoughts reflects dependencies on historical information, whereas states maintain answer-equivalence with the initial problem while progressively reducing execution complexity. *Figure (b)* illustrates differences in the number of prompt tokens and completion tokens for CoT, ToT, GoT, and the state-based AoT. For simplicity, we assume each thought consists of the same number of tokens, with an average of $O(n)$ thoughts required to express a solution. While ToT maintains b branches, resulting in a fixed number of b invocations per expansion stage, GoT's settings can be flexibly adjusted depending on the scenario and are thus denoted as $O(b)$.

problem, thereby significantly reducing historical dependencies. The reasoning process is expressed as a sequence of states with progressively reduced test-time complexity, rather than an accumulation of historical thoughts like CoT, as illustrated in Figure 1a. To ensure steady progress, we introduce a two-phase state transition mechanism: the decomposition stage converts the current state into a Directed Acyclic Graph (DAG)-based reasoning path, and the contraction stage uses its structure to reduce dependencies and generate the next state.

This fundamental structure with memoryless property distinguish our approach from many CoT-based methods. Thus, our method can be seamlessly integrated with existing test-time scaling methods, enhancing their scaling efficiency. While exploring integrations with tree search and reflective refinement to further scale up the Markovian reasoning chain, we identify an emergent trend towards an atomic reasoning structure (Figure 4), where reasoning trajectories are represented as a series of self-contained, low-complexity atomic problems. To emphasize this characteristic, we name our approach Atom of Thoughts (AoT).

Our contributions are summarized as follows:

- **Markovian Reasoning Process.** We introduce a general-purpose Markovian reasoning process that achieves high-quality and cost-effective reasoning across various scenarios, including code generation, mathematical reasoning, and multi-step reasoning tasks.
- **Scalable Reasoning Structure.** The basic structure design of Markov chain in AoT facilitates seamless integration with various test-time scaling methods, significantly enhancing computational efficiency and allowing the combination of different methods' advantages. This scalability ensures more effective utilization of increased computational budgets without the overhead of maintaining extensive historical contexts.
- **Atomic Reasoning.** Further leveraging AoT's seamless integration capability to enhance itself, by integrating with tree search and reflective refinement to scale up the exploration of the Markovian reasoning chain, we uncover an emergent atomic reasoning structure. In this structure, complex reasoning trajectories are decomposed into a sequence of atomic, self-

contained units with low complexity. This atomicization brings about improved reasoning performance and robustness.

2 Related Work

2.1 Reasoning Framework

Drawing inspiration from cognitive behaviors in human reasoning [3]—such as step-by-step decomposition [33, 57, 31, 13], reflective refinement [23, 55, 54], and aggregation ensemble [32, 20, 42]—various prompting strategies have been developed to enhance the reasoning capabilities of LLMs. These reasoning frameworks typically employ structured representations, including chains, graphs, and trees [40, 4, 51], to model the reasoning space efficiently and systematically. Chain-based methods, for instance, decompose complex problems into linear sequences of subproblems [33, 57, 31], primarily optimizing for stepwise dependency. In contrast, tree- and graph-based formalisms support hierarchical exploration of multiple reasoning paths, allowing for more dynamic adaptation during the problem-solving process [40, 4]. These structured approaches have demonstrably improved LLM performance in diverse applications like code generation, question answering, and complex data processing [17, 16, 48, 50], by enabling LLMs to tackle intricate problems with enhanced coherence and interpretability.

While these structured methods significantly expand LLMs' reasoning capabilities, they also inherently accumulate historical dependencies. This accumulation can lead to increased computational costs and potential interference during the inference process. Recent efforts have attempted to mitigate this reliance on historical information by exploring Markovian reasoning processes and atomic reasoning steps, aiming for more memoryless transitions [36, 34, 58, 35]. However, these approaches often suffer from task-specific design limitations, hindering generalizability and efficient parallelism [12, 38, 46]. In contrast, AoT introduces a DAG-based approach that decouples partial subproblems into atomic nodes. This decoupling enables independent state transitions without the substantial overhead associated with maintaining historical context. By iteratively decomposing problems into these atomic nodes and then contracting them, our method reduces overall complexity and inherently supports efficient parallel execution, thereby addressing the limitations of traditional chain, tree, and graph-based structures.

2.2 Test-Time Scaling

Test-time scaling has emerged as a powerful mechanism to enhance LLM reasoning by extending computational effort during inference. Framework-based approaches augment LLM capabilities through structured reasoning extensions, leveraging cognitive operations and external tool integration to facilitate deeper exploration of solution spaces [49, 28, 7]. These methods introduce reflective reasoning cycles, recursive problem-solving, and dynamic path selection, significantly improving performance on complex reasoning tasks. Despite these advances, existing techniques commonly preserve full historical state information throughout the reasoning process. This can lead to redundant computational overhead and potential conflicts across successive reasoning steps.

Recent work has explored alternative strategies, such as supervised fine-tuning on CoT trajectories, demonstrating improved LLM capacity to maintain coherent, long-term reasoning [44, 6, 41]. Reinforcement learning have further pushed these boundaries by enabling models to autonomously extend reasoning chains, potentially unlocking emergent cognitive patterns [22, 47, 9, 45]. However, similar to framework-based methods, these techniques often rely on maintaining expansive historical contexts, which can limit their efficiency and scalability as reasoning paths become extended.

In contrast to these history-dependent methods, our approach adopts a Markovian perspective, modeling the reasoning process as state transitions assisted by a temporary DAG structure. This memoryless design eliminates the need for redundant history tracking, focusing computational resources solely on current state transformations. Furthermore, our proposed two-phase transition mechanism, comprising decomposition and contraction stages, facilitates atomic problem-solving. This enhances computational efficiency while maintaining structural clarity. This structured yet flexible approach not only reduces dependency overhead but also aligns naturally with the principles of test-time scaling, offering seamless integration with existing reasoning frameworks to achieve scalable, high-performance inference.

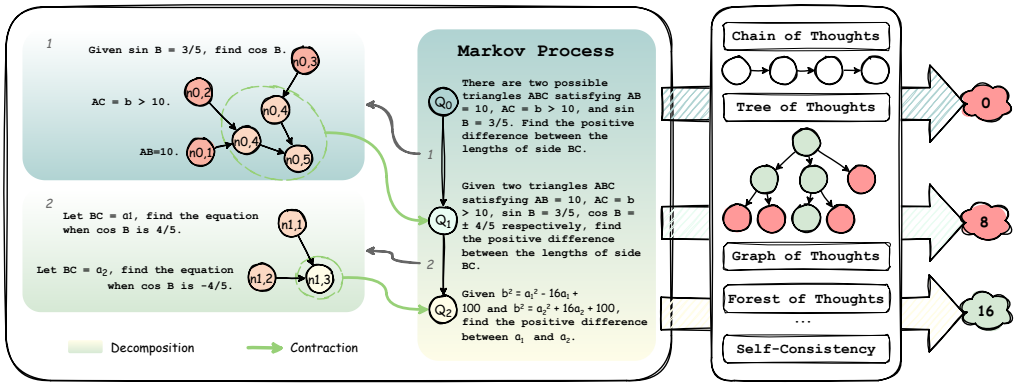


Figure 2: **Overview of AoT.** The Markov reasoning framework iteratively derives states Q_{i+1} from predecessors Q_i using DAG decomposition and contraction. The left part shows this iterative process, while the right part highlights the integration with existing methods. Any intermediate state Q_i can act as an entry point Q_0 for other methods, ensuring flexible composition while preserving answer equivalence to the original question. This allows AoT to operate independently or as a preprocessing module to optimize the performance or efficiency of existing approaches.

3 Atomic Reasoning via Markov Process

In this section, we first formally derive a Markovian reasoning process grounded in a clear probabilistic formulation. We then discuss how this Markovian reasoning structure can be integrated seamlessly with other reasoning methods to further scale up inference time. Finally, we demonstrate how atomic reasoning structures naturally emerge through such scaling-up procedures. The overview of this Markovian reasoning process is illustrated in Figure 2.

3.1 Markovian Reasoning Process

Reasoning Chain. CoT reasoning introduces a sequence of intermediate steps T_i to solve a problem. This process can be formalized as a probabilistic sampling procedure:

$$A \sim p(A|\mathcal{T}, Q_0) \prod_{i=0}^{N} p(T_i|\mathcal{T}_{<i}, Q_0) \tag{1}$$

where A is the final answer, and $\mathcal{T} = \{T_0, T_1, \ldots, T_N\}$ is the sequence of thoughts, each conditioned on the previous steps $\mathcal{T}_{<i}$ and the initial question Q_0.

An alternative formulation—Least-to-Most [57] prompting—reframes the node of chain as a sub-question Q_i, yielding:

$$A \sim p(A|\mathcal{Q}) \prod_{i=0}^{N} p(Q_i|\mathcal{Q}_{<i}) \tag{2}$$

Under the above formulation, the reasoning process is characterized by the accumulation of intermediate thoughts or subquestions in the sequence, leading to a continual increase in historical information. However, ideally, if the reasoning chain satisfies the property of a memoryless Markov process—where each state S_{i+1} depends only on S_i—we obtain:

$$A \sim p(A|S_N) \prod_{i=0}^{N} p(S_{i+1}|S_i) \tag{3}$$

where S_i represents a state in the Markovian reasoning process. In the following paragraph, we will explicitly clarify the semantic content of the Markov state S_i, resulting in a more specific and practical representation.

Markov State. In practice, real-world problems rarely satisfy the strict Markov assumption directly. To establish a meaningful Markovian formulation, we reuse the subquestion symbol Q_i to represent the Markov states S_i, initialized by the original question Q_0. Since the final answer A must be derivable from the final state Q_{-1}, it follows naturally that Q_{-1} is answer-equivalent to Q_0. Thus, an essential invariant emerges: each intermediate subquestion Q_i must preserve answer-equivalence with the original question. To ensure meaningful Markov state transitions, we further impose that the sequence of subquestions $\{Q_0, Q_1, \ldots, Q_N\}$ monotonically reduces in complexity, guaranteeing genuine reasoning progress at each transition.

Two-phase Transition However, state transitions aiming at test-time reduction remain challenging for LLMs, especially without task-specific training. This difficulty arises primarily from the complex historical dependencies within reasoning trajectories. To address this issue, we propose a two-phase transition mechanism that first explicitly decomposes the current state Q_i to capture the internal dependencies before contracting them into the next state.

In the decomposition phase, we introduce a DAG scaffold $\mathcal{G}_i$ to explicitly represent the dependency structure among reasoning steps within each intermediate question Q_i. This temporary structure is later discarded to eliminate historical dependencies, enabling the Markovian transition. Formally, the DAG is defined as:

$$\mathcal{G}_i = (\mathcal{N}, E), \quad E \subseteq \{(N_j, N_k) \mid j < k\} \tag{4}$$

where nodes N_k represent individual thoughts or subquestions, and edges (N_j, N_k) indicate that node N_j provides necessary information for node N_k.

In the subsequent contraction phase, we transform the temporary DAG structure $\mathcal{G}_i$ into the next Markov state Q_{i+1}. Specifically, nodes without incoming edges in $\mathcal{G}_i$ are independent and can be safely discarded, whereas the remaining dependent nodes are reformulated into an answer-equivalent independent question Q_{i+1}. Formally, the overall Markovian transition process can be expressed as:

$$A \sim p(A|Q_N) \prod_{i=0}^{N} p(Q_{i+1}|\mathcal{G}_i)\, p(\mathcal{G}_i|Q_i). \tag{5}$$

A detailed step-by-step example demonstrating the complete decomposition-contraction process is provided in Appendix B.2.

3.2 Emerged Atomic Reasoning

The Markovian reasoning process provides a fundamental, low-level structural prior for inference. In this subsection, we discuss the design of a termination mechanism to counteract the potential fragility introduced by strict memorylessness, thereby constructing a stable reasoning framework. Moreover, we describe how this Markovian reasoning structure can be combined with additional methods—particularly through structured exploration via tree search and reflective verification—to further scale up test-time reasoning. This combined approach reveals the emergence of a stable, indivisible reasoning structure, termed atomic reasoning.

Termination Strategy. Unlike CoT-based approaches, which can recover from early errors by leveraging accumulated context, our Markov chain lacks such a fallback due to its memoryless nature. This amplifies the risk of propagating low-quality transitions—if an intermediate question Q_{i+1} diverges semantically from the original task, subsequent reasoning becomes meaningless.

To address this, we introduce a quality-aware termination strategy. After each transition $Q_i \to Q_{i+1}$, an LLM-as-a-judge selects the best answer to the original question Q_0 from the triplet $\{\text{solve}(Q_i), \text{solve}(\mathcal{G}_i), \text{solve}(Q_{i+1})\}$. Crucially, this mechanism implicitly enforces answer equivalence: if Q_{i+1} fails to preserve answer equivalence with Q_0, then $\text{solve}(Q_{i+1})$ will not provide a valid answer for Q_0 and thus cannot be selected by the judge. This selection-based filtering naturally ensures that only semantically stable transformations maintaining answer equivalence are retained. If Q_{i+1} is not selected, the process terminates and returns the best candidate among the three. Detailed quality metrics demonstrating the effectiveness of this mechanism are provided in Appendix B.1.

Modular Integration. Since each Markov state is constrained to be an equivalently transformed representation of the original question, the reasoning process forms a semantically aligned and

Table 1: Performance Comparison.

Model	Benchmark	CoT	CoT-SC	SR	AR	AFlow	ToT	GoT	FoT	AoT
Non-Reasoning LLMs										
GPT-4o-mini	MATH	78.3	81.8	78.7	65.4	83.0	82.0	82.3	82.6	**83.6**
	GSM8K	90.9	92.0	91.7	87.2	93.5	91.8	92.1	94.2	**95.0**
	MBPP	72.4	73.2	72.8	70.1	74.0	73.5	73.7	74.8	**75.2**
	LongBench	57.6	58.6	58.2	52.9	61.0	59.0	59.2	60.8	**68.5**
DeepSeek-V3	MATH	94.4	95.2	94.8	90.1	96.1	95.0	95.3	95.6	**96.5**
	GSM8K	96.2	97.0	96.8	92.5	97.8	96.5	96.8	97.5	**98.2**
	MBPP	75.7	76.5	76.0	73.2	77.3	76.8	77.0	78.2	**79.6**
	LongBench	58.8	60.1	59.5	55.3	63.5	61.2	61.5	63.3	**71.0**
Reasoning LLMs										
O3-mini	AIME	79.6	81.0	80.2	76.0	82.5	81.2	81.5	81.8	**83.0**
	LiveCodeBench	23.6	25.0	24.2	20.0	26.5	25.2	25.5	27.8	**32.2**
	LongBench	56.3	57.5	56.8	52.0	58.0	56.5	56.8	57.9	**65.3**
DeepSeek-R1	AIME	78.3	79.7	78.9	74.7	81.2	79.9	80.2	80.5	**81.7**
	LiveCodeBench	24.5	25.9	25.1	20.9	27.4	26.1	26.4	28.1	**30.9**
	LongBench	55.1	56.2	55.4	52.3	58.7	57.0	57.5	58.2	**67.9**

fully self-contained sequence of problem representations. This property enables modular reasoning without compromising the integrity of the overall task. In practice, each state within the chain can be independently routed to specialized solvers, subjected to verification procedures, or further embedded into structured reasoning frameworks—such as tree-based or graph-based inference. The introduction of the Markov reasoning process thus does not merely offer an alternative to previous reasoning chain methods, but rather defines a structural foundation upon which diverse test-time reasoning strategies can be constructed.

Atomic Structure. Although the termination strategy ensures robustness, it also restricts the emergence of deeper reasoning chains. To explore the full potential of the Markov process, we sample and extend trajectories, combining tree search and reflection mechanisms. These structured explorations reveal a statistically supported phenomenon: deeper reasoning states tend to converge into irreducible forms, maintaining a stable and relatively low reasoning token count, from which the original problem's answer can be directly inferred with high execution stability. We refer to these stable forms as atomic structures: indivisible and self-contained representations that require no further decomposition. Importantly, atomicity is not imposed a priori, but emerges naturally as a property discovered throughout the reasoning process. This convergence toward atomic units represents a logical endpoint where problems become sufficiently simple that further decomposition is neither necessary nor beneficial. Notably, this convergence point is jointly determined by both the intrinsic complexity of the problem and the reasoning capabilities of the underlying model—different problems may converge at different depths, and the same problem may exhibit different atomic granularities when solved by models with varying capacities.

4 Experiments

Our experiments aim at two primary objectives. First, we conduct main experiments across a variety of datasets spanning mathematics, code generation, and multi-hop question answering to demonstrate the cost-efficiency advantages of AoT as a general-purpose reasoning framework. Second, leveraging the flexibility provided by the basic Markov chain structure in our approach, we design integration experiments at various granularities. These experiments explore the utilization of AoT as a plug-in component to enhance cost-efficiency in other reasoning frameworks and investigate scaling effects in integration with classical methods like tree search and verification-based reflection, analyzing emergent reasoning phenomena.

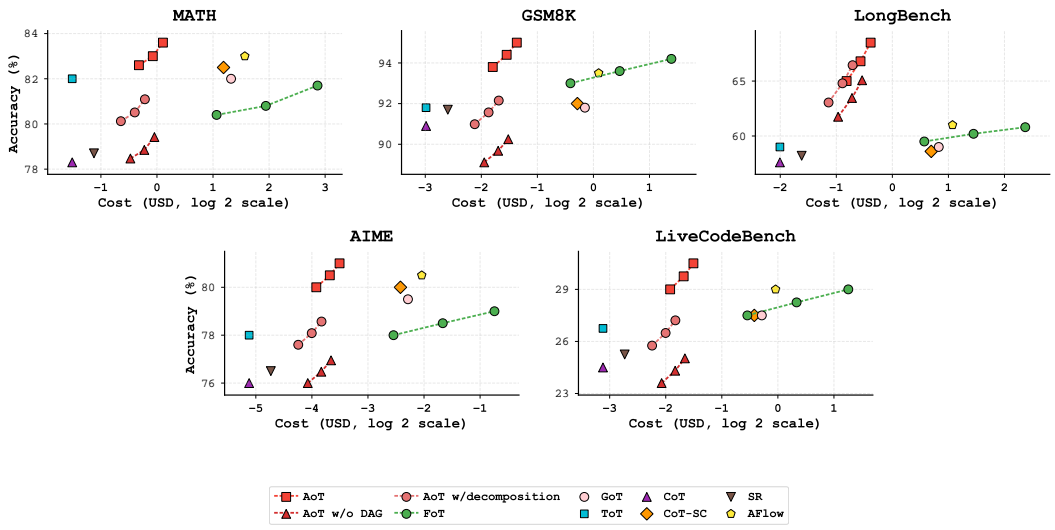


Figure 3: A comparison of performance and cost of various methods and ablation methods on the dataset, with GPT-4o-mini as the backbone. Each node in the curves represents an AoT (or ablation variants) iteration result, where increasing token consumption indicates deeper iterations. Due to relatively poor AR performance leading to scattered data points, AR data points are excluded.

4.1 Experimental Setup

Benchmarks and Metrics. We evaluate AoT across representative benchmarks covering mathematical reasoning (MATH [14], GSM8K [8], AIME[1]), code generation (MBPP [1], LiveCodeBench [19]), and multi-hop question answering tasks (HotpotQA [39], MuSiQue [30], and 2WikiMultiHopQA [15] preprocessed by LongBench [2]), see Appendix F.3 for details. Following previous work [49, 5], we report pass rates for mathematical and coding benchmarks, and F1 scores for multi-hop QA tasks.

Settings. All prompt templates used in Markov reasoning process for experiments are fully described in Appendix A.1. Key hyperparameters, including model temperature and Markov chain length, are detailed and discussed in Appendix A.2. We set the default temperature to 1.0 and the maximum Markov chain length to 3 for the main experiments to balance performance and efficiency while enabling scaling curves. Due to AoT 's design and termination mechanism, longer chain lengths increase the performance ceiling without linearly increasing costs.

Backbones and Baselines. AoT is designed to be compatible with various LLM backbones. To demonstrate its effectiveness, we employed two categories of LLMs. The first category comprises non-reasoning LLMs, specifically GPT-4o-mini [25] and DeepSeek-V3 [10]. The second category includes reasoning-capable LLMs such as O3-mini [27] and DeepSeek R1 [9]. Specifically, we use non-reasoning models to evaluate performance on MATH, GSM8K, and MBPP, and reasoning-capable models to evaluate performance on more challenging tasks such as AIME and LiveCodeBench. Additionally, since multi-hop QA is not a primary focus for reasoning-capable models, both categories of models are evaluated on LongBench for comprehensive comparison.

For comparison, we evaluated AoT against a diverse set of baseline methods, broadly categorized by their interaction pattern with the LLM: single-call or multi-call invocations. Single-call approaches include well-known techniques like Chain-of-Thought (CoT) [33] and Chain-of-Draft (CoD) [37]. Multi-call methods represent more complex workflows, such as CoT with Self-Consistency (CoT-SC) [32], Self-Refine (SR) [23], Analogical Prompting (AP) [43], Forest-of-Thought (FoT) [5], and the agentic framework AFlow [49]. Further details are provided in Appendix A.3.

[1] https://huggingface.co/datasets/Maxwell-Jia/AIME_2024

4.2 Main Results

Table 1 presents the main experimental results. Across both Non-Reasoning and Reasoning LLMs, AoT consistently demonstrates strong performance. For Non-Reasoning LLMs such as GPT-4o-mini and DeepSeek-V3, AoT achieves the highest scores on benchmarks like MATH, GSM8K, MBPP, and LongBench, often surpassing all other compared methods. For instance, with GPT-4o-mini, AoT scores 83.6 on MATH, 95.0 on GSM8K, 75.2 on MBPP, and 68.5 on LongBench, which are the top performances. Similarly, DeepSeek-V3 with AoT leads with scores on all benchmarks.

In the Reasoning LLMs section, featuring O3-mini and DeepSeek-R1, AoT continues to exhibit competitive and often leading performance. For O3-mini, AoT achieves the highest scores on AIME (83.0), LiveCodeBench (32.2), and LongBench (65.3). With DeepSeek-R1, AoT again leads on all tasks. Overall, AoT consistently achieves state-of-the-art or highly competitive results across a diverse set of models and benchmarks, demonstrating its effectiveness.

Figure 3 further demonstrates that performance improves progressively with additional reasoning iterations. This highlights the effectiveness of our proposed termination strategy: by mitigating error propagation from memoryless Markovian transitions, it preserves the desirable test-time scaling property—performance does not degrade as more computational resources are allocated.

4.3 Ablation Study

We conduct ablation studies to examine the impact of core components in our framework. Specifically, we evaluate two variants: (1) Without Decomposition, where the model directly contracts reasoning trajectories from the initial question without constructing a DAG; and (2) Without DAG-guided Contraction, where decomposition still occurs, but the contraction step does not rely on any structural guidance. In this setting, only the first naturally independent subproblem is separated out. Figure 3 shows that both ablations significantly degrade performance, with the second variant causing a more severe drop. This suggests that partial or superficial structural cues can be more harmful than providing none at all. These results underscore the importance of explicitly modeling fine-grained dependencies in reasoning trajectories, showing that faithful structural representations meaningfully enhance reasoning effectiveness and precision. Comprehensive quality metrics for the DAG generation process, including answer equivalence maintenance rates (>99% across all datasets) and complexity reduction rates (74-82%), are provided in Appendix B.1.

4.4 Scaling Up Analysis

In this section, we further explore the scalability of AoT by integrating it with existing reasoning frameworks, leveraging its flexible, modular design. We begin our analysis by using individual Markov states as integration points—a lightweight and straightforward approach where intermediate states processed by AoT serve as optimized entry points for other reasoning methods. Our experiments reveal substantial efficiency improvements at test-time, which encourages us to examine larger, more structured integration granularities to fully capitalize on the structural strengths of our framework. Notably, as we progressively extend the Markov chain during scaling analysis, we observe a consistent reduction in the number of tokens required for reasoning in the final states. Through detailed analysis, we identify emerging atomic characteristics in the reasoning trajectories, motivating us to design further scaling-up experiments based on this property.

State Integration. The Markov states Q_i generated by AoT represent simplified, yet answer-equivalent reformulations of the original questions, making them ideal entry points for external methods. Indeed, AoT itself demonstrates such modular integration potential, employing basic CoT-style prompting to solve each intermediate state. To experimentally validate the effectiveness of these intermediate states, we investigate whether initiating reasoning using optimized intermediate states Q_1 can enhance both accuracy and computational efficiency in external frameworks. The results, illustrated in Figure 4, confirm that starting reasoning from these optimized intermediate states notably improves performance while simultaneously reducing computational costs, as demonstrated in the integration with frameworks such as FoT.

Tree Searching. Beyond single-state integration, the full Markov sequence $\mathcal{Q}$ generated by AoT can provide a structured scaffold for more complex reasoning frameworks, effectively replacing

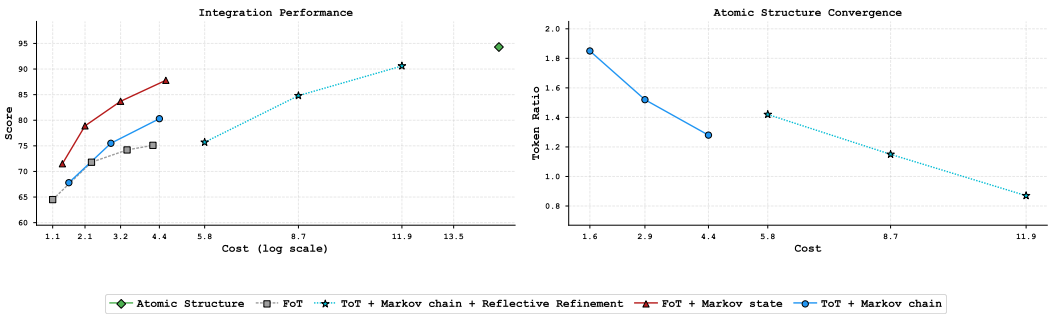


Figure 4: The process involves gradually enhancing integration for scaling up at test time. ToT uses three branches, while FoT employs two, four, and eight trees, respectively.

traditional CoT-based structures. In conventional CoT-based ToT, the inherent randomness of LLM-based sampling can lead to inconsistencies in reasoning chain lengths, causing nodes at the same depth to represent varying stages of reasoning progress. This inconsistency complicates node comparison and diminishes pruning effectiveness. In contrast, the Markov chains constructed by AoT ensure answer equivalence between each intermediate node and the original question, thereby guaranteeing fundamental comparability across nodes at the same depth. This structural consistency significantly enhances the gains from scaling through parallel sampling at test-time.

Reflective Refinement. Termination strategy in AoT provides a safeguard for the quality of single-pass Markov reasoning. When a transition yields a low-quality intermediate state, early termination allows the system to avoid wasting computation on unpromising paths. However, this conservative mechanism may also limit further exploration. To address this, we augment our method with verification-based reflection, where transitions $Q_i \rightarrow Q_{i+1}$ are evaluated by an LLM-as-a-judge to assess whether the newly generated state exhibits a significant degradation in test-time performance. If such degradation is detected, the system triggers a reflective refinement step, encouraging deeper and more meaningful reasoning rather than trivial reformulations. This reflective verification substantially improves comparability between nodes at the same depth, increases the effective exploration space, and further amplifies the benefits of structural scaling. When combining all three integration strategies (ToT + Markov chain + Reflective Refinement), we observe significant performance gains: for instance, on MATH, this full integration achieves 84.9% accuracy compared to ToT's 82.0%, and on AIME, it reaches 81.2% versus ToT's 78.0%, demonstrating the compounding benefits of our modular design.

Atomic Struture. Due to the inherent scalability of the AoT architecture, deeper Markov chains—enabled by both tree search and verification-based reflection—exhibit stronger test-time performance and require fewer reasoning tokens in the final state. Statistical analysis reveals that the token count of final reasoning steps gradually approaches that of a minimal DAG representation comprising all independent subproblems generated during transitions. This suggests a natural convergence toward atomic states—questions that are semantically represent indivisible reasoning units. We refer to this phenomenon as atomic reasoning, where the entire reasoning trajectory is composed of such minimal, non-decomposable elements. To further validate this insight, we conduct an additional experiment where we isolate and re-execute these highly atomic reasoning paths independently. While this incurs significantly higher computational cost, the results exhibit stable scaling trends, highlighting the structural advantages of AoT with high budget.

5 Conclusions and Future Work

We present AoT, a general-purpose reasoning framework that leverages Markovian transitions to minimize historical dependencies during inference. By alternating between decomposition and contraction, AoT incrementally reduces complex queries into atomic subproblems, enabling scalable and modular reasoning across maths, code, and multi-hop QA tasks. Empirically, we show that AoT not only scales gracefully with compute but also integrates flexibly into existing reasoning paradigms as a plug-in module. Limitations and broader impacts of AoT are provided in Appendix **??** and **??**.

While AoT offers a promising path toward atomic reasoning, its current implementation operates solely at inference time. A natural extension is to align this structure with training-time objectives—teaching models to internalize Markovian and atomic reasoning patterns directly. This could involve supervised fine-tuning with synthetic traces, reinforcement learning over decomposition trajectories, or pretraining on datasets that promote context-isolated reasoning.

More broadly, this work lays the foundation for reasoning systems that emphasize minimal context, compositionality, and structural modularity. We hope AoT serves as a stepping stone toward more efficient, interpretable, and robust reasoning with large language models.

References

[1] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[2] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench: A bilingual, multitask benchmark for long context understanding. In *ACL (1)*, pages 3119–3137. Association for Computational Linguistics, 2024.

[3] W. Bechtel, A. Abrahamsen, and G. Graham. Cognitive science: History. In Neil J. Smelser and Paul B. Baltes, editors, *International Encyclopedia of the Social & Behavioral Sciences*, pages 2154–2158. Pergamon, Oxford, 2001. ISBN 978-0-08-043076-8. doi: https://doi.org/10.1016/B0-08-043076-7/01442-X. URL `https://www.sciencedirect.com/science/article/pii/B008043076701442X`.

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

[5] Zhenni Bi, Kai Han, Chuanjian Liu, Yehui Tang, and Yunhe Wang. Forest-of-thought: Scaling test-time compute for enhancing LLM reasoning. *CoRR*, abs/2412.09078, 2024.

[6] Edward Y. Chang, Yuxuan Tong, Morry Niu, Graham Neubig, and Xiang Yue. Demystifying long chain-of-thought reasoning in llms. *CoRR*, abs/2502.03373, 2025.

[7] Lingjiao Chen, Jared Quincy Davis, Boris Hanin, Peter Bailis, Ion Stoica, Matei Zaharia, and James Zou. Are more LLM calls all you need? towards scaling laws of compound inference systems. *CoRR*, abs/2403.02419, 2024.

[8] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168, 2021.

[9] DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL `https://arxiv.org/abs/2501.12948`.

[10] DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou,

Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, and Wangding Zeng. Deepseek-v3 technical report. *CoRR*, abs/2412.19437, 2024. doi: 10.48550/ARXIV.2412.19437. URL https://doi.org/10.48550/arXiv.2412.19437.

[11] Ruomeng Ding, Chaoyun Zhang, Lu Wang, Yong Xu, Minghua Ma, Wei Zhang, Si Qin, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. Everything of thoughts: Defying the law of penrose triangle for thought generation. In *ACL (Findings)*, pages 1638–1662. Association for Computational Linguistics, 2024.

[12] 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. In *EMNLP*, pages 8154–8173. Association for Computational Linguistics, 2023.

[13] Shibo Hao, Yi Gu, Haotian Luo, Tianyang Liu, Xiyan Shao, Xinyuan Wang, Shuhua Xie, Haodi Ma, Adithya Samavedhi, Qiyue Gao, Zhen Wang, and Zhiting Hu. LLM reasoners: New evaluation, library, and analysis of step-by-step reasoning with large language models. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=b0y6fbSUG0.

[14] 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. In *NeurIPS Datasets and Benchmarks*, 2021.

[15] Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing A multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *COLING*, pages 6609–6625. International Committee on Computational Linguistics, 2020.

[16] Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhao Wu, Ceyao Zhang, Chenxing Wei, Danyang Li, Jiaqi Chen, Jiayi Zhang, et al. Data interpreter: An llm agent for data science. *arXiv preprint arXiv:2402.18679*, 2024.

[17] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for A multi-agent collaborative framework. In *ICLR*. OpenReview.net, 2024.

[18] Zhenyu Hou, Xin Lv, Rui Lu, Jiajie Zhang, Yujiang Li, Zijun Yao, Juanzi Li, Jie Tang, and Yuxiao Dong. Advancing language model reasoning through reinforcement learning and inference scaling. *CoRR*, abs/2501.11651, 2025.

[19] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2024.

[20] Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. Llm-blender: Ensembling large language models with pairwise ranking and generative fusion. In *ACL (1)*, pages 14165–14178. Association for Computational Linguistics, 2023.

[21] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020.

[22] Kimi, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*, 2025.

[23] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *NeurIPS*, 2023.

[24] Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel J. Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. *CoRR*, abs/2501.19393, 2025.

[25] OpenAI. Gpt-4o mini: advancing cost-efficient intelligence, 2024. URL `https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/`.

[26] OpenAI. Introducing openai o1, 2024. URL `https://openai.com/o1/`.

[27] OpenAI. OpenAI o3-mini: Pushing the frontier of cost-effective reasoning, 2025. URL `https://openai.com/index/openai-o3-mini/`.

[28] Jon Saad-Falcon, Adrian Gamarra Lafuente, Shlok Natarajan, Nahum Maru, Hristo Todorov, Etash Guha, Estefany Kelly Buchanan, Mayee F. Chen, Neel Guha, Christopher Ré, and Azalia Mirhoseini. Archon: An architecture search framework for inference-time techniques. *CoRR*, abs/2409.15254, 2024.

[29] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *CoRR*, abs/2408.03314, 2024.

[30] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multihop questions via single-hop question composition. *Trans. Assoc. Comput. Linguistics*, 10:539–554, 2022.

[31] Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *ACL (1)*, pages 2609–2634. Association for Computational Linguistics, 2023.

[32] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *ICLR*. OpenReview.net, 2023.

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

[34] Kun Xiang, Zhili Liu, Zihao Jiang, Yunshuang Nie, Runhui Huang, Haoxiang Fan, Hanhui Li, Weiran Huang, Yihan Zeng, Jianhua Han, Lanqing Hong, Hang Xu, and Xiaodan Liang. Atomthink: A slow thinking framework for multimodal mathematical reasoning. *CoRR*, abs/2411.11930, 2024.

[35] Kun Xiang, Zhili Liu, Zihao Jiang, Yunshuang Nie, Kaixin Cai, Yiyang Yin, Runhui Huang, Haoxiang Fan, Hanhui Li, Weiran Huang, et al. Can atomic step decomposition enhance the self-structured reasoning of multimodal large models? *arXiv preprint arXiv:2503.06252*, 2025.

[36] Amy Xin, Jinxin Liu, Zijun Yao, Zhicheng Lee, Shulin Cao, Lei Hou, and Juanzi Li. Atomr: Atomic operator-empowered large language models for heterogeneous knowledge reasoning. *CoRR*, abs/2411.16495, 2024.

[37] Silei Xu, Wenhao Xie, Lingxiao Zhao, and Pengcheng He. Chain of draft: Thinking faster by writing less. *CoRR*, abs/2502.18600, 2025. doi: 10.48550/ARXIV.2502.18600. URL `https://doi.org/10.48550/arXiv.2502.18600`.

[38] Wen Yang, Kai Fan, and Minpeng Liao. Markov chain of thought for efficient mathematical reasoning. *CoRR*, abs/2410.17635, 2024.

[39] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *EMNLP*, pages 2369–2380. Association for Computational Linguistics, 2018.

[40] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *NeurIPS*, 2023.

[41] Xinhao Yao, Ruifeng Ren, Yun Liao, and Yong Liu. Unveiling the mechanisms of explicit cot training: How chain-of-thought enhances reasoning generalization. *CoRR*, abs/2502.04667, 2025.

[42] Yuxuan Yao, Han Wu, Mingyang LIU, Sichun Luo, Xiongwei Han, Jie Liu, Zhijiang Guo, and Linqi Song. Determine-then-ensemble: Necessity of top-k union for large language model ensembling. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=FDnZFpHmU4.

[43] Michihiro Yasunaga, Xinyun Chen, Yujia Li, Panupong Pasupat, Jure Leskovec, Percy Liang, Ed H. Chi, and Denny Zhou. Large language models as analogical reasoners. In *ICLR*. OpenReview.net, 2024.

[44] Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. Limo: Less is more for reasoning. *arXiv preprint arXiv:2502.03387*, 2025.

[45] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

[46] Oussama Zekri, Ambroise Odonnat, Abdelhakim Benechehab, Linus Bleistein, Nicolas Boullé, and Ievgen Redko. Large language models as markov chains. *CoRR*, abs/2410.02724, 2024.

[47] Weihao Zeng, Yuzhen Huang, Qian Liu, Wei Liu, Keqing He, Zejun Ma, and Junxian He. Simplerl-zoo: Investigating and taming zero reinforcement learning for open base models in the wild. *arXiv preprint arXiv:2503.18892*, 2025.

[48] Guibin Zhang, Kaijie Chen, Guancheng Wan, Heng Chang, Hong Cheng, Kun Wang, Shuyue Hu, and Lei Bai. Evoflow: Evolving diverse agentic workflows on the fly. *CoRR*, abs/2502.07373, 2025.

[49] Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. *arXiv preprint arXiv:2410.10762*, 2024.

[50] Jiayi Zhang, Chuang Zhao, Yihan Zhao, Zhaoyang Yu, Ming He, and Jianping Fan. Mobileexperts: A dynamic tool-enabled agent team in mobile devices. *arXiv preprint arXiv:2407.03913*, 2024.

[51] Jinghan Zhang and Kunpeng Liu. Thought space explorer: Navigating and expanding thought space for large language model reasoning. In *2024 IEEE International Conference on Big Data (BigData)*, pages 8259–8251. IEEE, 2024.

[52] Yifan Zhang, Yang Yuan, and Andrew Chi-Chih Yao. On the diagram of thought. *CoRR*, abs/2409.10038, 2024.

[53] Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. Automatic chain of thought prompting in large language models. In *ICLR*. OpenReview.net, 2023.

[54] Chuanyang Zheng, Zhengying Liu, Enze Xie, Zhenguo Li, and Yu Li. Progressive-hint prompting improves reasoning in large language models. *CoRR*, abs/2304.09797, 2023.

[55] Huaixiu Steven Zheng, Swaroop Mishra, Xinyun Chen, Heng-Tze Cheng, Ed H. Chi, Quoc V. Le, and Denny Zhou. Take a step back: Evoking reasoning via abstraction in large language models. In *ICLR*. OpenReview.net, 2024.

[56] Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *ICML*. OpenReview.net, 2024.

[57] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V. Le, and Ed H. Chi. Least-to-most prompting enables complex reasoning in large language models. In *ICLR*. OpenReview.net, 2023.

[58] Pei Zhou, Jay Pujara, Xiang Ren, Xinyun Chen, Heng-Tze Cheng, Quoc V. Le, Ed H. Chi, Denny Zhou, Swaroop Mishra, and Huaixiu Steven Zheng. SELF-DISCOVER: large language models self-compose reasoning structures. In *NeurIPS*, 2024.

Appendix Overview

This appendix is organized into three main parts: Section A provides comprehensive implementation details including prompts, hyperparameters, and baseline configurations; Section B presents detailed empirical analyses validating our framework's effectiveness; and Sections **??–??** discuss limitations and broader impacts of this work.

A Implementation Details

This section provides comprehensive implementation details necessary for reproducing our experiments, including prompt templates, hyperparameter settings, and baseline method configurations.

A.1 Prompt Templates

We present the core prompt structures used in AoT for different task domains. Our framework employs four key prompt types: (1) `direct` for solving problems, (2) `decompose` for extracting DAG structures, (3) `contract` for generating simplified questions, and (4) `judge` for LLM-as-a-judge evaluation. Below we detail domain-specific implementations for mathematical reasoning, code generation, and multi-hop question answering.

Design Rationale. The Multi-hop QA prompts use JSON for structured responses, while Math and Code tasks use HTML-like tags (e.g., `<answer></answer>`). This design choice reflects task-specific requirements: JSON naturally accommodates Multi-hop QA's need for structured outputs including reasoning chains and supporting evidence, while HTML tags provide clear answer demarcation for Math and Code tasks. Function parameters also vary by domain—Multi-hop QA requires context passages, Code generation needs test cases and dependency information, while Math tasks only require the question. These variations align with the inherent characteristics of each problem type rather than representing arbitrary design choices.

A.1.1 Mathematical Reasoning

```
def direct(question: str):
    instruction = """
        You are a precise math question solver. Solve the given math
    question step by step using a standard algebraic approach:

        QUESTION: {question}

        You can freely reason in your response, but please enclose the
     final answer within <answer></answer> tags (pure number without
    units and explanations)
    """
    prompt = instruction.format(question=question)
    return prompt

def decompose():
    instruction = """
        Decompose the previous reasoning trajectory into a series of
    sub-questions or thoughts.

        Instructions:
        1. Each sub-question or thought should list its other sub-
    questions or thoughts' indexes it depends (0-based, can be an
    empty list)
```

```
        2. Dependencies are defined as information needed in sub-
    question or thought that:
            - Does NOT come directly from the original question
            - MUST come from previous sub-questions or thoughts
    """
    return instruction

def contract():
    instruction = """
        Generate a simplified intermediate form of the original
    question based on the previous sub-questions or thoughts step by
    step.

        The previous sub-questions or thoughts with marked
    dependencies actually form a directed acyclic graph (DAG), where
    nodes whose dependencies is empty list can be regarded as
    independent sub-questions or thoughts.

        The simplified question must be:
        1. self-contained: The simplified question's description must
    contain all information needed to solve itself, without requiring
    additional information from the original question or reasoning
    trajectory
        2. test-time reduced: The simplified question must require
    fewer reasoning steps compared to the original question (these
    steps are reduced because these solved independent sub-problems or
     thoughts become known conditions in the simplified question or
    excluded as incorrect explorations)

    """
    formatter = "Last step, enclose the question within <question></
    question> tags"
    instruction += formatter
    return instruction

def judge(question: str, solutions: list):
    instruction = """
        Here is the original problem:
        {question}

        Here are some reference solutions:
        {solutions}

        Ensemble the best answer to the original problem from the
    solutions step by step:
    """
    formatter = "Last step, enclose the answer within <answer></answer
    > tags (must be an integer or decimal number without units and
    explanations)"
    instruction += formatter

    solutions_str = ""
    for i, solution in enumerate(solutions):
        solutions_str += f"solution {i}: {solution}\n"
    prompt = instruction.format(question=question, solutions=
    solutions_str)
    return prompt
```

Listing 1: Math

A.1.2 Code Generation

```
def direct(question: str, contexts: str):
```

```
    instruction = """
        Solve the following problem step by step:
        {question}
        Your code should be a python function with format: {contexts}

        Please extend your reasoning process as much as possible; the
    longer the chain of thought, the better.

    """
    formatter = "Last step, enclose your code within ```python and ```
    "
    instruction += formatter
    prompt = instruction.format(question=question, contexts=contexts)
    return prompt

def decompose():
    instruction = """
        Decompose the previous reasoning trajectory into a series of
    sub-questions or thoughts.

        Instructions:
        1. Each sub-question or thought should list its other sub-
    questions or thoughts' indexes it depends (0-based, can be an
    empty list)
        2. Dependencies are defined as information needed in sub-
    question or thought that:
            - Does NOT come directly from the original question
            - MUST come from previous sub-questions or thoughts
    """
    return instruction

def contract(dag, test_cases):
    instruction = """
        Generate a simplified intermediate form of the original
    problem based on the variable dependency analysis.

        You ast.arg given a directed acyclic graph (DAG) representing
    the dependencies between variables in the original code:
        {dag}

        And the original test cases:
        {test_cases}

        The simplified problem must be:
        1. Self-contained: The description must contain all
    information needed to solve itself, without requiring additional
    context from the original problem
        2. Test-time reduced: The simplified problem must require
    fewer reasoning steps by using intermediate variables from the
    original code as direct inputs

        Your task is to:
        1. Create a simplified version of the problem that starts with
     intermediate variables as inputs
        2. Generate new test cases that use these intermediate
    variables as parameters while maintaining the exact same expected
    outputs as in the original test cases

        Do not use any code examples in your simplified problem
    formulation.
    """
    formatter = r"Enclose the simplified problem within <question></
    question> tag and the new test cases (assert codes, use \n to
    split each case) within <test></test> tag"
    instruction += formatter
```

```
    prompt = instruction.format(dag=dag, test_cases=test_cases)
    return prompt

def judge(question: str, solutions: list):
    instruction = """
        Here is the original problem:
        {question}

        Here are some reference solutions:
        {solutions}

        Give the index of the best solution as your answer.
    """
    formatter = "Last step, enclose the answer within <answer></answer
    > tags (0-based)"
    instruction += formatter

    solutions_str = ""
    for i, solution in enumerate(solutions):
        solutions_str += f"solution {i}: {solution}\n"
    prompt = instruction.format(question=question, solutions=
    solutions_str)
    return prompt
```

Listing 2: Code

A.1.3 Multi-hop Question Answering

```
def direct(question: str, contexts: str):
    instruction = """
        Solve the following multi-hop question step by step:
        {question}

        CONTEXTS:
        {contexts}

        Firstly, you need to extract the relevant supporting sentences
     from the original text, then cut out the continuous segments as
    the answer.
    """
        formatter = """
    Provide your response in this JSON format:
    {{
        "question": {question},
        "thought": "give your step by step thought process here",
        "supporting_sentences": [
            "Include ALL sentences needed to justify your answer",
            "Use ... for long sentences when appropriate"
        ],
        "answer": "Your precise answer following the instructions
    above" or "none" if no answer can be found
    }}
    """
    instruction += formatter
    prompt = instruction.format(question=question, contexts=contexts)
    return prompt

def decompose(question: str, trajectory: str, answer: str):
    instruction = """
        You are tasked with breaking down a multiple choice question
    reasoning process into sub-questions.

        Original Question: {question}
```

```
        Complete Reasoning Process: {trajectory}

        Instructions:
        1. Break down the reasoning process into a series of sub-
    questions
        2. Each sub-question should:
           - Be written in interrogative form
           - Have a clear answer
           - List its other sub-questions' indexes it depends (0-based
    , can be an empty list)
        3. Dependencies are defined as information needed to answer
    the current sub-question that:
           - Does NOT come directly from the original question
           - MUST come from the answers of previous sub-questions
    """
    formatter = """
        Format your response as the following JSON object:
        {{
            "thought": "<the thought process of how to step by step
    propose the sub-questions until the answer of the original
    question in the given reasoning process is obtained>",
            "sub-questions": [
                {{
                    "description": "<the description of the sub-
    question>",
                    "answer": <the answer to the sub-question>,
                    "depend": [<indices of the dependent sub-questions
    >, ...]
                }}
            ],
            "answer": "{answer}"
        }}
    """
    return (instruction + formatter).format(question=question,
    trajectory=trajectory, answer=answer)

def contract(question: str, decompose_result: dict, independent: list,
     dependent: list):
    instruction = """
        You are a multiple choice question solver specializing in
    optimizing step-by-step reasoning processes. Your task is to
    optimize the existing reasoning trajectory into a more efficient,
    single self-contained question.

        For the original question: {question}

        Here are step-by-step reasoning process:
        {response}

        {sub_questions}

        Here are explanations of key concepts:
        1. self-contained: The optimized question must be solvable
    independently, without relying on any external information
        2. efficient: The optimized question must be simpler than the
    original, requiring fewer reasoning steps and having a clearer
    reasoning process (these steps are reduced because some solved sub
    -problems become known conditions in the optimized question or are
     excluded as incorrect explorations)

        Note: Since this is a multiple choice question, the optimized
    question must completely retain the options of the original
    question.
```

```
        You can freely reason in your response, but please enclose the
     your optimized question within <question></question> tags
    """
    sub_questions = """
        The following sub-questions and their answers can serve as
    known conditions:
        {independent}

        The descriptions of the following questions can be used to
    form the description of the optimized problem:
        {dependent}

        """
    answer = decompose_result["answer"]
    for sub_q in independent:
        sub_q.pop("depend", None)
    for sub_q in dependent:
        sub_q.pop("depend", None)

    sub_questions = sub_questions.format(independent=independent,
    dependent=dependent)
    return instruction.format(question=question, answer=answer,
    response=decompose_result["response"], sub_questions=sub_questions
    )

def judge(question: str, solutions: list):
    instruction = """
        You are a precise multiple choice question solver. Compare
    then synthesize the best answer from multiple solutions to select
    the most correct option:

        QUESTION: {question}

        SOLUTIONS:
        {solutions}

        Extend your chain of thought as much as possible; the longer
    the chain of thought, the better.

        You can freely reason in your response, even propose new
    reasoning to get a better answer than all solutions, but please
    mark the final option with <answer>single letter of your chosen
    option</answer> tags
    """

    solutions_str = ""
    for i, solution in enumerate(solutions):
        solutions_str += f"solution {i}: {solution}\n"
    prompt = instruction.format(question=question, solutions=
    solutions_str)
    return prompt
```

Listing 3: Multi-hop QA

A.2 Hyperparameter Configuration

Maximum Transition Count. The maximum number of transitions in the Markovian reasoning chain is a key hyperparameter that controls the depth of reasoning exploration. Theoretically, longer chains enable deeper reasoning, but practical considerations require balancing performance gains with computational efficiency. Throughout our experiments, we uniformly set the maximum transition count to 3, which empirically provides an effective trade-off (see Section B.3 for empirical justification based on structural depth analysis).

Adaptive Setting. For query-specific optimization, the maximum transition count can be dynamically determined by analyzing the initial DAG structure. Since each transition ideally eliminates one layer of independent nodes (those without incoming edges), the depth of the initially decomposed DAG $\mathcal{G}_0$ serves as a reasonable upper bound estimate for the required number of transitions. This can be computed via a simple graph traversal without additional LLM invocations.

Other Hyperparameters. We use temperature $T = 1.0$ for all LLM sampling operations to balance exploration and determinism. For integration experiments with tree-based methods (Section 4.4), we use 3 branches for ToT and vary the number of trees in FoT as $\{2, 4, 8\}$ to study scaling behavior.

A.3 Baseline Implementation Details

This subsection describes our implementation of baseline methods to ensure fair and reproducible comparisons.

A.3.1 Forest of Thoughts (FoT)

In our implementation, we utilize the classical Tree of Thoughts (ToT) approach as the fundamental tree structure within the Forest of Thoughts framework, while maintaining several critical mechanisms from the original FoT design, including majority voting for aggregating results across different trees and expert evaluation for assessing solution quality.

However, our implementation differs from the original FoT in certain aspects to accommodate a broader range of question types. Specifically, we remove the early stopping criteria that terminate tree splitting when nodes cannot produce valid outputs. While this mechanism is particularly effective for constrained tasks like Game-of-24 where rule-based validation is straightforward, it is less applicable to our diverse evaluation scenarios where output validity is less clearly defined. Instead, we maintain tree expansion regardless of intermediate output quality, allowing the framework to explore potentially valuable paths that might initially appear suboptimal. Additionally, we omit the Input Data Augmentation technique, as analogical reasoning approaches do not demonstrate consistent effectiveness across different question domains in our experiments.

These modifications preserve the core strengths of FoT while enhancing its adaptability to a wider range of reasoning tasks. Our implementation successfully reproduces the scaling curves reported in the original FoT paper and achieves superior performance across multiple benchmarks.

A.3.2 AFlow

For AFlow, we adopt the optimal workflows identified in the original work for each benchmark dataset while making necessary adaptations to our experimental setup. For mathematical reasoning tasks on MATH and GSM8K, we directly employ AFlow's proven optimal workflows. For multi-hop reasoning scenarios in LongBench, we use the workflow initially optimized for HotpotQA, as both datasets share core multi-hop reasoning characteristics. This approach ensures we leverage AFlow's strengths while maintaining consistency across similar problem types.

A.3.3 Dataset-Specific Details

For the MATH dataset, we filter out questions with non-integer or non-decimal answers to ensure consistent evaluation. We evaluate the first 1,000 cases from MATH for efficiency, while assessing the remaining benchmarks in their entirety.

B Empirical Analysis and Validation

This section presents detailed empirical analyses that validate the effectiveness of our framework, including quality metrics for DAG generation, concrete examples of the decomposition-contraction process, and statistical analyses of structural properties.

B.1 DAG Generation Quality Assessment

To evaluate the quality of our two-phase transition mechanism (decomposition and contraction), we provide comprehensive quality metrics across multiple datasets. Table 2 presents three key metrics that assess different aspects of the DAG generation and state transition process.

Table 2: DAG Generation Quality Metrics Across Benchmarks

Metric	MATH	GSM8K	MBPP	LongBench
Answer Equivalence Maintenance	99.2%	99.5%	99.7%	99.3%
Test-time Complexity Reduction	76.4%	82.1%	74.8%	79.2%
LLM-as-a-Judge Selection Rate	92.5%	95.8%	83.1%	91.5%

Evaluation Methodology. Both answer equivalence and test-time complexity reduction are assessed through LLM evaluation, where the evaluator LLM is provided with Q_i and Q_{i+1} along with their execution processes. The LLM judges answer equivalence by examining whether the reasoning trajectory's derivation goals remain consistent, and assesses complexity reduction by analyzing the trajectory length and required reasoning steps.

Metric Definitions.

- **Answer Equivalence Maintenance**: The probability that the contracted question Q_{i+1} maintains answer equivalence with the original question Q_0. The consistently high rates (>99% across all datasets) demonstrate the reliability of our transition mechanism.
- **Test-time Complexity Reduction**: The probability that Q_{i+1} successfully reduces reasoning complexity compared to Q_i. Rates between 74%-82% indicate that most transitions achieve meaningful simplification.
- **LLM-as-a-Judge Selection Rate**: The probability that Q_{i+1} is selected (or at least not rejected) by the LLM-as-a-judge when choosing among $\{\text{solve}(Q_i), \text{solve}(\mathcal{G}_i), \text{solve}(Q_{i+1})\}$. High selection rates (83%-96%) confirm that the generated states provide valuable reasoning improvements.

These metrics collectively demonstrate that our DAG generation and contraction process reliably maintains answer equivalence while achieving meaningful complexity reduction, validating the effectiveness of our two-phase transition design.

B.2 Detailed DAG Decomposition and Contraction Examples

To provide concrete understanding of how the DAG decomposition and contraction mechanism works in practice, we present detailed step-by-step examples demonstrating the complete two-phase transition process.

B.2.1 Mathematical Reasoning Example

Original Problem (Q_0): *There are two possible triangles ABC satisfying AB = 10, AC = b > 10, and sin B = 3/5. Find the positive difference between the lengths of side BC.*

Step 1: Direct Reasoning Trajectory. When solving Q_0 directly, the LLM generates the following reasoning chain:

1. Given sin B = 3/5, we can find cos B using the Pythagorean identity: $\cos^2 B + \sin^2 B = 1$
2. This gives us $\cos^2 B = 1 - (3/5)^2 = 1 - 9/25 = 16/25$
3. Therefore, cos B = ±4/5 (two possible values)
4. Using the Law of Cosines: $b^2 = a^2 + c^2 - 2ac\cos B$, where $a = BC$, $c = AB = 10$
5. For cos B = 4/5: $b^2 = a^2 + 100 - 20a(4/5) = a^2 - 16a + 100$
6. For cos B = -4/5: $b^2 = a^2 + 100 - 20a(-4/5) = a^2 + 16a + 100$

7. Solving these two cases yields two possible values for BC
8. The positive difference is computed from these two values

Step 2: DAG Decomposition ($Q_0 \to \mathcal{G}_0$). The LLM decomposes this reasoning trajectory into a dependency structure:

Node 0: "Calculate cos B from sin B = 3/5 using the Pythagorean identity"

- Dependencies: [] (no dependencies, independent subproblem)
- Result: cos B = ±4/5

Node 1: "Given AB = 10, AC = b > 10, and cos B = ±4/5, apply the Law of Cosines to find the two possible values of BC"

- Dependencies: [0] (depends on the result of Node 0)

Node 2: "Calculate the positive difference between the two values of BC"

- Dependencies: [1] (depends on the result of Node 1)

The DAG structure is: Node 0 $\to$ Node 1 $\to$ Node 2, forming a linear chain of depth 3.

Step 3: Contraction ($\mathcal{G}_0 \to Q_1$). Nodes without incoming edges (Node 0) represent independent subproblems that can be directly solved. After solving Node 0, we obtain cos B = ±4/5. This information is incorporated into the problem statement, and nodes depending on it are reformulated:

Contracted Question (Q_1): *Given that cos B can be either 4/5 or -4/5, with AB = 10 and AC = b > 10, use the Law of Cosines to find the two possible values of BC, then calculate their positive difference.*

Key observations:

- Q_1 is self-contained: All necessary information (cos B values) is now explicitly stated
- Q_1 maintains answer equivalence with Q_0: Solving Q_1 yields the same final answer
- Q_1 has reduced test-time complexity: The trigonometric calculation is eliminated, reducing reasoning steps from 8 to approximately 5
- The DAG depth is reduced from 3 to 2 (only Nodes 1 and 2 remain)

Step 4: LLM-as-a-Judge Selection. After generating the triplet $\{\text{solve}(Q_0), \text{solve}(\mathcal{G}_0), \text{solve}(Q_1)\}$, the LLM-as-a-judge evaluates which provides the best answer to the original problem Q_0. In this case:

- solve(Q_0): Direct solution with full reasoning chain
- solve$(\mathcal{G}_0)$: Solution by explicitly solving each node in the DAG
- solve(Q_1): Solution of the contracted problem

If Q_1 maintains answer equivalence (which it does), solve(Q_1) will provide a valid answer and is likely to be selected due to its cleaner reasoning structure. If the contraction process had failed to maintain equivalence, solve(Q_1) would produce an incorrect or nonsensical answer, and the judge would select one of the other options, naturally filtering out the failed transition.

Iteration Potential. If we continue from Q_1, a second transition could further decompose and contract the problem, potentially separating the two Law of Cosines calculations from the difference computation. This iterative process continues until reaching an atomic state where no further meaningful decomposition is possible.

B.2.2 Key Insights from the Example

This example illustrates several important aspects of our framework:

1. **Structural Guidance:** The DAG explicitly captures dependencies, allowing the contraction phase to identify which information can be "baked into" the problem statement (Node 0's result) versus which must remain as reasoning steps (Nodes 1-2).
2. **Answer Equivalence:** The contracted question Q_1 asks for exactly the same final answer as Q_0, ensuring the Markov property holds while making meaningful progress.
3. **Complexity Reduction:** By solving independent subproblems and incorporating their results, Q_1 requires fewer reasoning steps, reducing the test-time computational burden.
4. **Implicit Quality Control:** The LLM-as-a-judge mechanism naturally filters failed transitions—if contraction produces an invalid or non-equivalent question, it won't be selected, preventing error propagation.

B.3 Analysis of Structural Diversity

To understand the structural characteristics of problems decomposed by our framework and provide empirical justification for our hyperparameter choices, we analyze the DAG structures generated from the first 1,000 questions of the MATH dataset.

B.3.1 Graph Structure and Chain Length

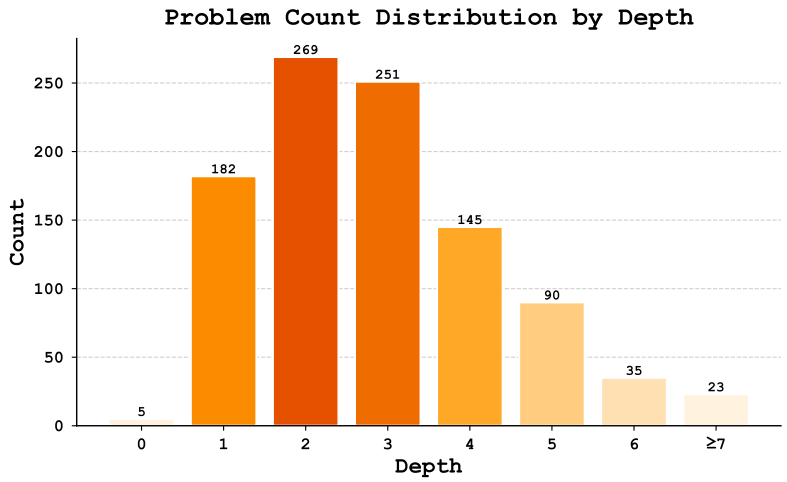


Figure 5: Distribution of solution depths across questions. Darker orange bars indicate depths that appear more frequently in the dataset.

Figures 5 and 6 reveal clear structural patterns in the decomposed questions. The depth distribution (Figure 5) shows that most questions exhibit depths between 2 and 4, with depth 3 being the most common pattern. This observation provides empirical justification for our choice of maximum transition count (3) in the main experiments—the structural depth naturally aligns with the transition requirements for most problems.

Similarly, the subquestion count distribution (Figure 6) indicates that questions typically decompose into 2 to 5 subquestions, with 3-4 subquestions representing the most frequent pattern. These statistics suggest that most reasoning problems naturally decompose into a small number of manageable subproblems, supporting our framework's design assumption that complex reasoning can be effectively simplified through structured decomposition.

B.3.2 Correlation Between Structural Complexity and Performance

Notably, we observed correlations between these structural metrics and solution accuracy. The scatter plots reveal two important patterns: First, as shown in Figure 8, as the depth of the solution graph increases, there is a general trend of decreasing accuracy. Second, as illustrated in Figure 7, questions with more subquestions tend to show lower accuracy rates. The color intensity of the points provides additional insight - darker points represent more common structural patterns in our dataset, showing

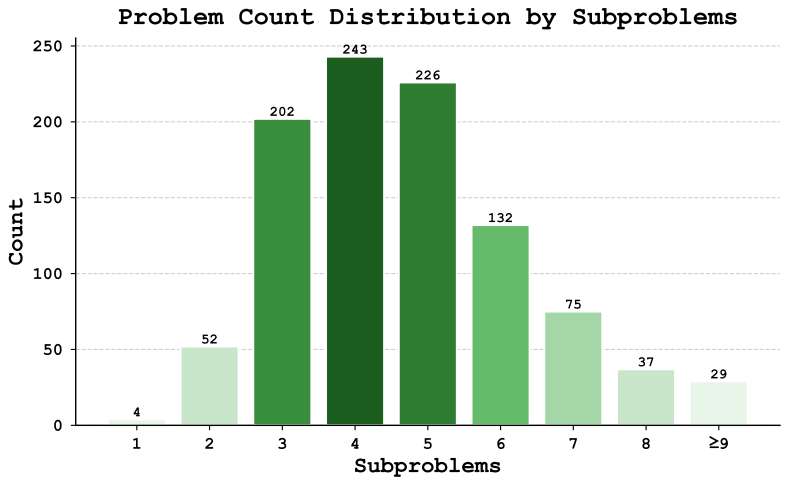


Figure 6: Distribution of subquestion counts across questions. Darker green bars represent more common subquestion counts in the solutions.

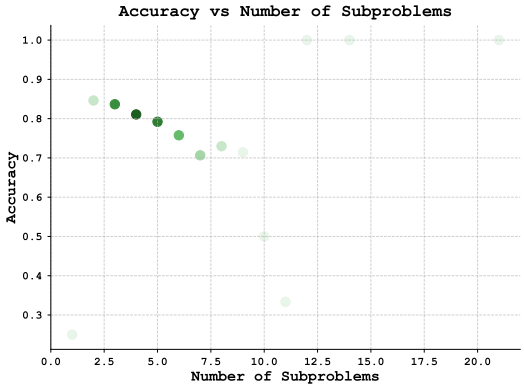


Figure 7: Number of subquestions vs accuracy. Color intensity (green) reflects data density - darker points represent more frequent patterns.

that most of our high-accuracy solutions come from questions with moderate depth and subquestion counts. This suggests that more complex question structures, characterized by either greater depth or more subquestions, pose greater challenges for question-solving systems. The decline in accuracy could be attributed to error propagation through longer solution chains and the increased cognitive load required to maintain consistency across more complex question structures.

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes] , [No] , or [NA] .
- [NA] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.

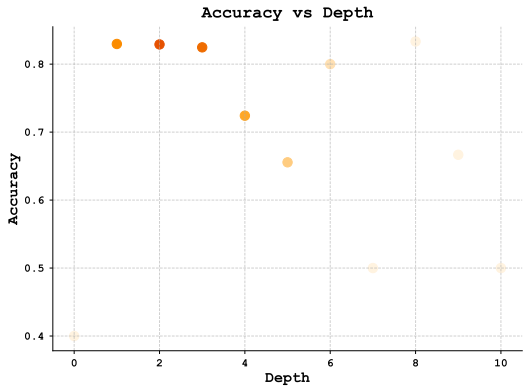


Figure 8: Solution depth vs accuracy. Color intensity (orange) reflects data density - darker points represent more frequent patterns.

- Please provide a short (1–2 sentence) justification right after your answer (even for NA).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will be asked to also include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While "[Yes] " is generally preferable to "[No] ", it is perfectly acceptable to answer "[No] " provided a proper justification is given (e.g., "error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering "[No] " or "[NA] " is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

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

   Answer: [Yes]

   Justification: Our claims in the abstract and introduction are precise and consistent with our findings.

   Guidelines:

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

2. **Limitations**

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

Answer: [Yes]

Justification: We discuss limitations in the Appendix **??**.

Guidelines:

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

3. **Theory assumptions and proofs**

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

   Answer: [Yes]

   Justification: For all the propositions and theorems presented in this paper, we provide mostly self-contained proofs in Section 3. For certain parts of the proofs, we refer to well-established results from recognized papers in the literature.

   Guidelines:

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

4. **Experimental result reproducibility**

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

   Answer: [Yes]

Justification: We provide detailed descriptions of our experimental setup in Section 4. We submit the code with the paper and will also release the source code to further facilitate reproducibility.

Guidelines:

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

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

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

   Answer: [Yes]

   Justification: We have provided the code in the supplementary file. The datasets we use are publicly available and have been accessed with the necessary permissions.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.

- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: We provide a dedicated section in Section 4 that details our experimental setups.

   Guidelines:

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

7. **Experiment statistical significance**

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

   Answer: [Yes]

   Justification: Experimental results are tested multiple times to ensure stability and reliability.

   Guidelines:

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

8. **Experiments compute resources**

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

   Answer: [Yes]

Justification: We provide comprehensive details about the compute resources used in both the experimental setup and results sections, ensuring reproducibility.

Guidelines:

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

9. **Code of ethics**

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

Answer: [Yes]

Justification: The research conducted in this paper fully conforms to the NeurIPS Code of Ethics in every respect.

Guidelines:

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

10. **Broader impacts**

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

Answer: [Yes]

Justification: We discuss both the potential positive and negative societal impacts of our work in Appendix **??**.

Guidelines:

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

11. **Safeguards**

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

    Answer: [NA]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

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

    Answer: [Yes]

    Justification: The paper cites the original paper that produced the models or datasets.

    Guidelines:

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

13. **New assets**

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

    Answer: [Yes]

    Justification: We provide the documentations alongside with the submitted code for reproducibility.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.

- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

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

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

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

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

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

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

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

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

16. **Declaration of LLM usage**

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

    Answer: [NA]

    Justification: The LLM is only used for writing and not as part of the core methods.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.