arXiv:2506.06941v1 [cs.AI] 7 Jun 2025

The Illusion of Thinking: Understanding the Strengths and Limitations of Reasoning Models via the Lens of Problem Complexity

Parshin Shojaee*† Iman Mirzadeh* Keivan Alizadeh
Maxwell Horton Samy Bengio Mehrdad Farajtabar

Apple

Abstract

Recent generations of frontier language models have introduced Large Reasoning Models (LRMs) that generate detailed thinking processes before providing answers. While these models demonstrate improved performance on reasoning benchmarks, their fundamental capabilities, scaling properties, and limitations remain insufficiently understood. Current evaluations primarily focus on established mathematical and coding benchmarks, emphasizing final answer accuracy. However, this evaluation paradigm often suffers from data contamination and does not provide insights into the reasoning traces' structure and quality. In this work, we systematically investigate these gaps with the help of controllable puzzle environments that allow precise manipulation of compositional complexity while maintaining consistent logical structures. This setup enables the analysis of not only final answers but also the internal reasoning traces, offering insights into how LRMs "think". Through extensive experimentation across diverse puzzles, we show that frontier LRMs face a complete accuracy collapse beyond certain complexities. Moreover, they exhibit a counter-intuitive scaling limit: their reasoning effort increases with problem complexity up to a point, then declines despite having an adequate token budget. By comparing LRMs with their standard LLM counterparts under equivalent inference compute, we identify three performance regimes: (1) low-complexity tasks where standard models surprisingly outperform LRMs, (2) medium-complexity tasks where additional thinking in LRMs demonstrates advantage, and (3) high-complexity tasks where both models experience complete collapse. We found that LRMs have limitations in exact computation: they fail to use explicit algorithms and reason inconsistently across puzzles. We also investigate the reasoning traces in more depth, studying the patterns of explored solutions and analyzing the models' computational behavior, shedding light on their strengths, limitations, and ultimately raising crucial questions about their true reasoning capabilities.

1 Introduction

Large Language Models (LLMs) have recently evolved to include specialized variants explicitly designed for reasoning tasks—Large Reasoning Models (LRMs) such as OpenAI's o1/o3 [1, 2], DeepSeek-R1 [3], Claude 3.7 Sonnet Thinking [4], and Gemini Thinking [5]. These models are new artifacts, characterized by their "*thinking*" mechanisms such as long Chain-of-Thought (CoT) with self-reflection, and have demonstrated promising results across various reasoning benchmarks. Their

*Equal contribution.
†Work done during an internship at Apple.
{p_shojaee, imirzadeh, kalizadehvahid, mchorton, bengio, farajtabar}@apple.com

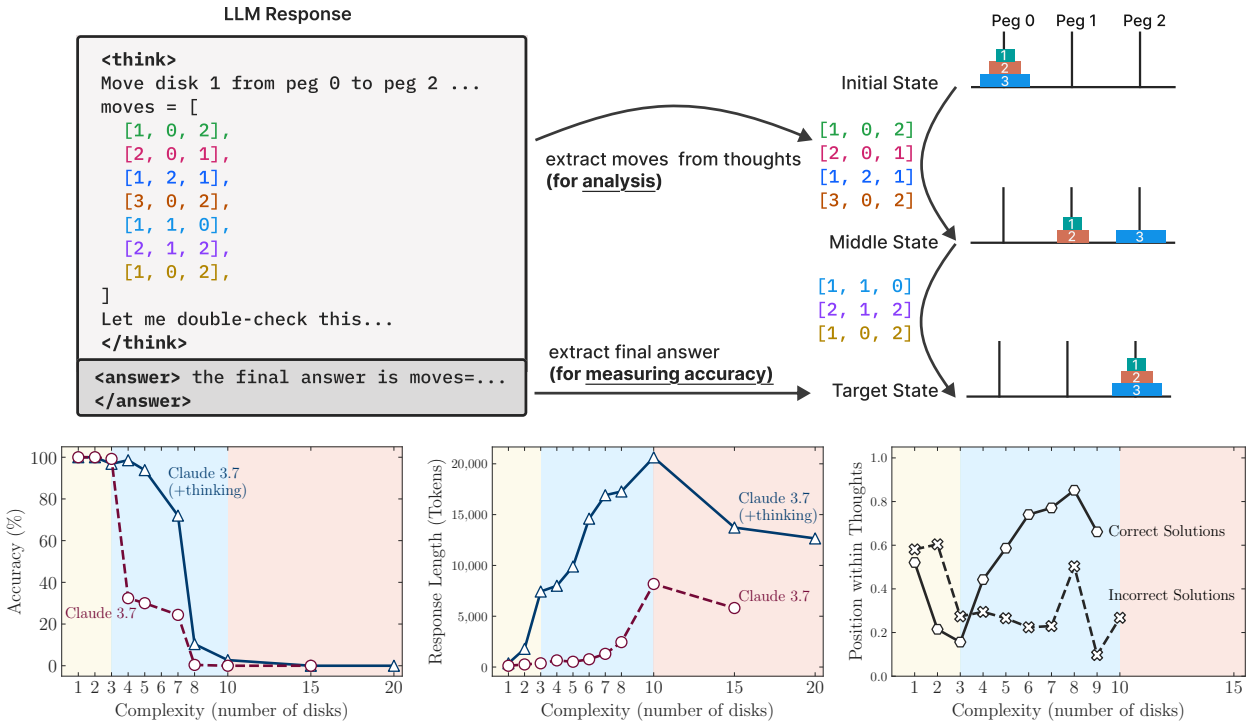


Figure 1: **Top**: Our setup enables verification of both final answers and intermediate reasoning traces, allowing detailed analysis of model thinking behavior. **Bottom left & middle**: At low complexity, non-thinking models are more accurate and token-efficient. As complexity increases, reasoning models outperform but require more tokens—until both collapse beyond a critical threshold, with shorter traces. **Bottom right**: For correctly solved cases, Claude 3.7 Thinking tends to find answers early at low complexity and later at higher complexity. In failed cases, it often fixates on an early wrong answer, wasting the remaining token budget. Both cases reveal inefficiencies in the reasoning process.

emergence suggests a potential paradigm shift in how LLM systems approach complex reasoning and problem-solving tasks, with some researchers proposing them as significant steps toward more general artificial intelligence capabilities.

Despite these claims and performance advancements, the fundamental benefits and limitations of LRMs remain insufficiently understood. Critical questions still persist: Are these models capable of generalizable reasoning, or are they leveraging different forms of pattern matching [6]? How does their performance scale with increasing problem complexity? How do they compare to their non-thinking standard LLM counterparts when provided with the same inference token compute? Most importantly, what are the inherent limitations of current reasoning approaches, and what improvements might be necessary to advance toward more robust reasoning capabilities?

We believe the lack of systematic analyses investigating these questions is due to limitations in current evaluation paradigms. Existing evaluations predominantly focus on established mathematical and coding benchmarks, which, while valuable, often suffer from data contamination issues and do not allow for controlled experimental conditions across different settings and complexities. Moreover, these evaluations do not provide insights into the structure and quality of reasoning traces. To understand the reasoning behavior of these models more rigorously, we need environments that enable controlled experimentation.

In this study, we probe the reasoning mechanisms of frontier LRMs through the lens of problem

complexity. Rather than standard benchmarks (e.g., math problems), we adopt controllable puzzle environments that let us vary complexity systematically—by adjusting puzzle elements while preserving the core logic—and inspect both solutions and internal reasoning (Fig. 1, top). These puzzles: (1) offer fine-grained control over complexity; (2) avoid contamination common in established benchmarks; (3) require only the explicitly provided rules, emphasizing algorithmic reasoning; and (4) support rigorous, simulator-based evaluation, enabling precise solution checks and detailed failure analyses.
Our empirical investigation reveals several key findings about current Language Reasoning Models (LRMs): First, despite their sophisticated self-reflection mechanisms learned through reinforcement learning, these models fail to develop generalizable problem-solving capabilities for planning tasks, with performance collapsing to zero beyond a certain complexity threshold. Second, our comparison between LRMs and standard LLMs under equivalent inference compute reveals three distinct reasoning regimes (Fig. 1, bottom). For simpler, low-compositional problems, standard LLMs demonstrate greater efficiency and accuracy. As problem complexity moderately increases, thinking models gain an advantage. However, when problems reach high complexity with longer compositional depth, both model types experience complete performance collapse (Fig. 1, bottom left). Notably, near this collapse point, LRMs begin reducing their reasoning effort (measured by inference-time tokens) as problem complexity increases, despite operating well below generation length limits (Fig. 1, bottom middle). This suggests a fundamental inference time scaling limitation in LRMs' reasoning capabilities relative to problem complexity. Finally, our analysis of intermediate reasoning traces or thoughts reveals complexity-dependent patterns: In simpler problems, reasoning models often identify correct solutions early but inefficiently continue exploring incorrect alternatives—an "overthinking" phenomenon. At moderate complexity, correct solutions emerge only after extensive exploration of incorrect paths. Beyond a certain complexity threshold, models completely fail to find correct solutions (Fig. 1, bottom right). This indicates LRMs possess limited self-correction capabilities that, while valuable, reveal fundamental inefficiencies and clear scaling limitations.
These findings highlight both the strengths and limitations of existing LRMs, raising questions about the nature of reasoning in these systems with important implications for their design and deployment. Our key contributions are:

- We question the current evaluation paradigm of LRMs on established math benchmarks and design a controlled experimental testbed by leveraging algorithmic puzzle environments that enable controllable experimentation with respect to problem complexity.
- We show that state-of-the-art LRMs (e.g., o3-mini, DeepSeek-R1, Claude-3.7-Sonnet-Thinking) still fail to develop generalizable problem-solving capabilities, with accuracy ultimately collapsing to zero beyond certain complexities across different environments.
- We find that there exists a scaling limit in the LRMs' reasoning effort with respect to problem complexity, evidenced by the counterintuitive decreasing trend in the thinking tokens after a complexity point.
- We question the current evaluation paradigm based on final accuracy and extend our evaluation to intermediate solutions of thinking traces with the help of deterministic puzzle simulators. Our analysis reveals that as problem complexity increases, correct solutions systematically emerge at later positions in thinking compared to incorrect ones, providing quantitative insights into the self-correction mechanisms within LRMs.
- We uncover surprising limitations in LRMs' ability to perform exact computation, including their failure to benefit from explicit algorithms and their inconsistent reasoning across puzzle types.

2 Related Works

Reasoning in Language Models. Large Language Models (LLMs) undergo multiple costly training phases using vast amounts of training data. While these LLMs demonstrate promising language understanding with strong compression capabilities, their intelligence and reasoning abilities remain a critical topic of scientific debate [7, 8]. Earlier iterations of LLMs [9, 10, 11] exhibited poor performance on reasoning benchmarks [12, 13, 14, 6]. To address these shortcomings, several approaches have been explored with the common theme among them being *"scaling"* both the training data and test-time computation. For instance, generating a Chain of Thought (CoT) [15, 16, 17, 18] and incorporating self-verification [19, 20, 21] prior to the final answer have been shown to improve model performance. However, obtaining high-quality and scalable CoT data is quite expensive due to its scarcity. Another line of research focuses on compensating for the lack of supervised data by teaching models to think more effectively through supervised learning or reinforcement learning [22, 23, 24, 25, 26, 27]. A notable open-source example of these improvements is Deepseek-R1 [3], which demonstrated that applying RL with verifiable rewards can significantly enhance model performance, matching that of closed models like OpenAI's o1 [2], leading to a new generation of language models referred to as Large Reasoning Models (LRMs) such as Gemini flash thinking [5], Claude 3.7 Sonnet thinking [4], etc.

Understanding Large Reasoning Models. Recent studies have explored various aspects of reasoning behavior: Large Reasoning Models have shown emergent behaviors such as discrepancy between thought traces and final answers [28, 29] as well as efficiency concerns through what researchers term the *"overthinking phenomenon"* [30, 31, 32, 33], where models produce verbose, redundant outputs, even after finding the solution, creating significant inference computational overhead. In this work, we systematically analyze how much model thinks w.r.t task complexity. Recently, Ballon et al. [34] demonstrated that in newer LRMs accuracy generally declines when thinking increases in math problems, in contrast we observe when in controlled puzzle environment difficulty passes a certain level the model starts to think less and opposite corelation of thinking and task complexity only happens up to some threshold. Yue et al. [35] questioned whether reinforcement learning truly elicits novel reasoning patterns and shows pass@k of reasoning vs non-reasoning models converge to the same point. We also observe that in MATH-500 pass@k is close for reasoning versus non-reasoning models but we observed different patterns under medium and high complexity of puzzles, which is not easily observable on established math benchmarks used in common evaluations.

Controllable Evaluation Environments. Unlike earlier studies that focused on mathematical problems to evaluate the reasoning capabilities of language models, this work introduces controllable puzzle environments. These environments allow for precise manipulation of problem complexity while maintaining consistent logical processes, enabling a more rigorous analysis of reasoning patterns and limitations. Controllable environments are not uncommon in the literature [12, 36, 37]. However, our primary aim is not to propose a new benchmark; instead, we use these benchmarks as tools for designing experiments to understand the reasoning capabilities of language models. A closely related study by Valmeekam et al. [38] demonstrated that o1-models show significant performance improvements compared to previous models. Our work offers additional insights, such as examining pairs of thinking/non-thinking models (e.g., DeepSeek-R1/V3, Claude 3.7 Sonnet thinking/non-thinking). Furthermore, we study the reasoning traces of the LRMs in more depth, revealing different behaviors across various complexity levels.

Overall, the promising results from recent LRMs raise a critical question: how much have the previously reported limitations of LLMs been improved? In this work, we move beyond merely measuring the performance of these LRMs. We analyze how well these LRMs tackle problems of varying complexities and examine the properties of their reasoning processes.

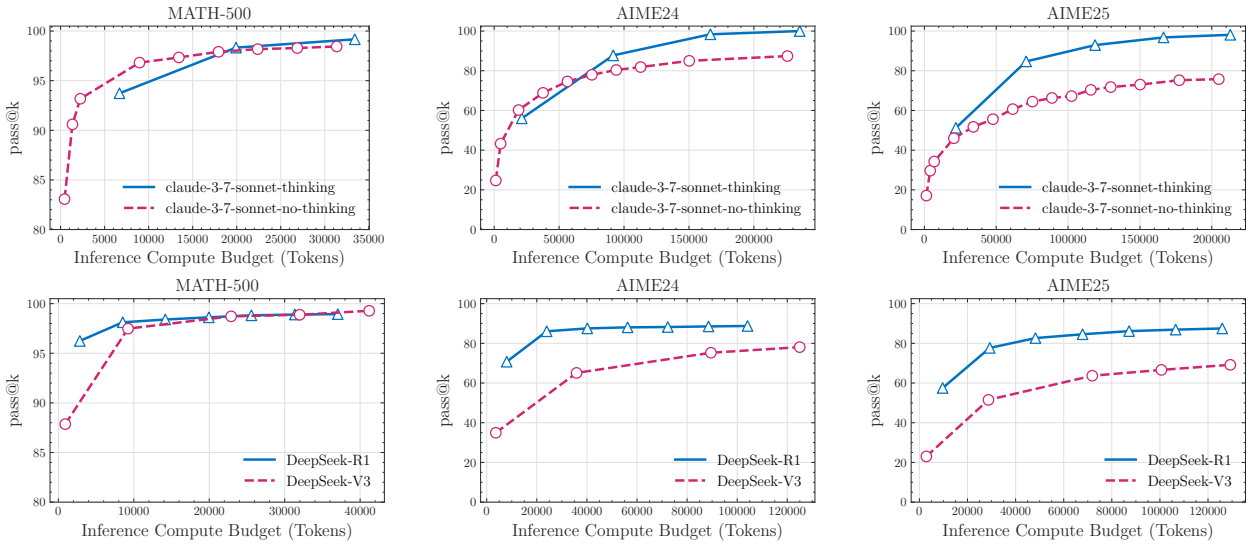


Figure 2: Comparative analysis of thinking versus non-thinking models across math benchmarks reveals inconsistent performance patterns. While results on the MATH-500 dataset show comparable performance between both model types, the thinking models demonstrate superior performance on AIME24 and AIME25 benchmarks. Additionally, the observed performance degradation from AIME24 to AIME25 highlights the vulnerability of these benchmarks to data contamination issues.

3 Math and Puzzle Environments

Currently, it is not clear whether the performance enhancements observed in recent RL-based thinking models are attributable to increased exposure to established mathematical benchmark data, to the significantly greater inference compute allocated to thinking tokens, or to reasoning capabilities developed by RL-based training? Recent studies [35, 39] have explored this question with established math benchmarks by comparing the upper-bound capabilities (pass@k) of RL-based thinking models with their non-thinking standard LLM counterparts. They have shown that under equivalent inference token budgets, non-thinking LLMs can eventually reach performance comparable to thinking models on benchmarks like MATH500 [40] and AIME24 [41]. We also conducted our comparative analysis of frontier LRMs like *Claude-3.7-Sonnet (with vs. without thinking)* and *DeepSeek (R1 vs. V3)*. Our results (shown in Fig. 2) confirm that, on the MATH500 dataset, the pass@k performance of thinking models is comparable to their non-thinking counterparts when provided with the same inference token budget. However, we observed that this performance gap widens on the AIME24 benchmark and widens further on AIME25. This widening gap presents an interpretive challenge. It could be attributed to either: (1) increasing complexity requiring more sophisticated reasoning processes, thus revealing genuine advantages of the thinking models for more complex problems, or (2) reduced data contamination in newer benchmarks (particularly AIME25). Interestingly, human performance on AIME25 was actually higher than on AIME24 [42, 43], suggesting that AIME25 might be less complex. Yet models perform worse on AIME25 than AIME24—potentially suggesting data contamination during the training of frontier LRMs. Given these non-justified observations and the fact that mathematical benchmarks do not allow for controlled manipulation of problem complexity, we turned to puzzle environments that enable more precise and systematic experimentation.

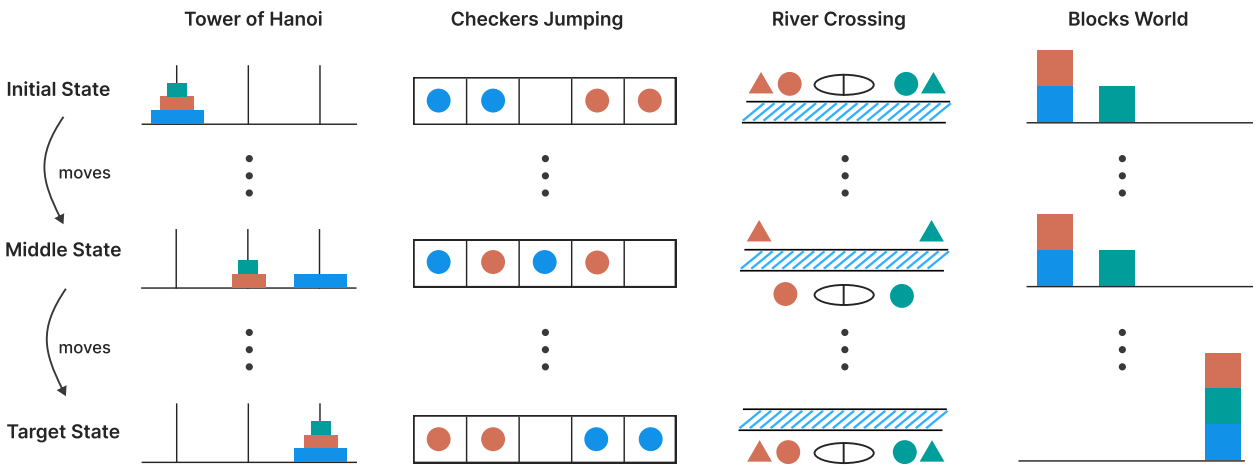


Figure 3: Illustration of the four puzzle environments. Columns show the progression from **initial state (top)** through **intermediate state (middle)** to **target state (bottom)** for puzzles: Tower of Hanoi (disk transfer across pegs), Checkers Jumping (position swapping of colored tokens), River Crossing (transporting entities across a river), and Blocks World (stack reconfiguration).

3.1 Puzzle Environments

We evaluate LRM reasoning on four controllable puzzles spanning compositional depth, planning complexity, and distributional settings. The puzzles are defined below and illustrated in Fig. 3.

Tower of Hanoi is a puzzle featuring three pegs and n disks of different sizes stacked on the first peg in size order (largest at bottom). The goal is to transfer all disks from the first peg to the third peg. Valid moves include moving only one disk at a time, taking only the top disk from a peg, and never placing a larger disk on top of a smaller one. The difficulty in this task can be controlled by the number of initial disks as the minimum number of required moves with n initial disks will be $2^n - 1$. However, in this work we do not grade for optimality of final solution and only measuring the correctness of each move and reaching the target state.

Checker Jumping is a one-dimensional puzzle arranging red checkers, blue checkers, and a single empty space in a line. The objective is to swap the positions of all red and blue checkers, effectively mirroring the initial configuration. Valid moves include sliding a checker into an adjacent empty space or jumping over exactly one checker of the opposite color to land in an empty space. No checker can move backward in the puzzle process. The complexity of this task can be controlled by the number of checkers: with $2n$ checkers, the minimum number of moves required will be $(n+1)^2 - 1$.

River Crossing is a constraint satisfaction planning puzzle involving n actors and their corresponding n agents who must cross a river using a boat. The goal is to transport all $2n$ individuals from the left bank to the right bank. The boat can carry at most k individuals and cannot travel empty. Invalid situations arise when an actor is in the presence of another agent without their own agent present, as each agent must protect their client from competing agents. The complexity of this task can also be controlled by the number of actor/agent pairs present. For $n = 2, n = 3$ pairs, we use boat capacity of $k = 2$ and for larger number of pairs we use $k = 3$.

Blocks World is a block-stacking puzzle requiring rearrangement of blocks from an initial configuration into a specified goal configuration. The objective is to find the minimum number of moves needed for this transformation. Valid moves are restricted to the topmost block of any stack, which can be placed either on an empty stack or on top of another block. The complexity in this task can be controlled by the number of blocks present.

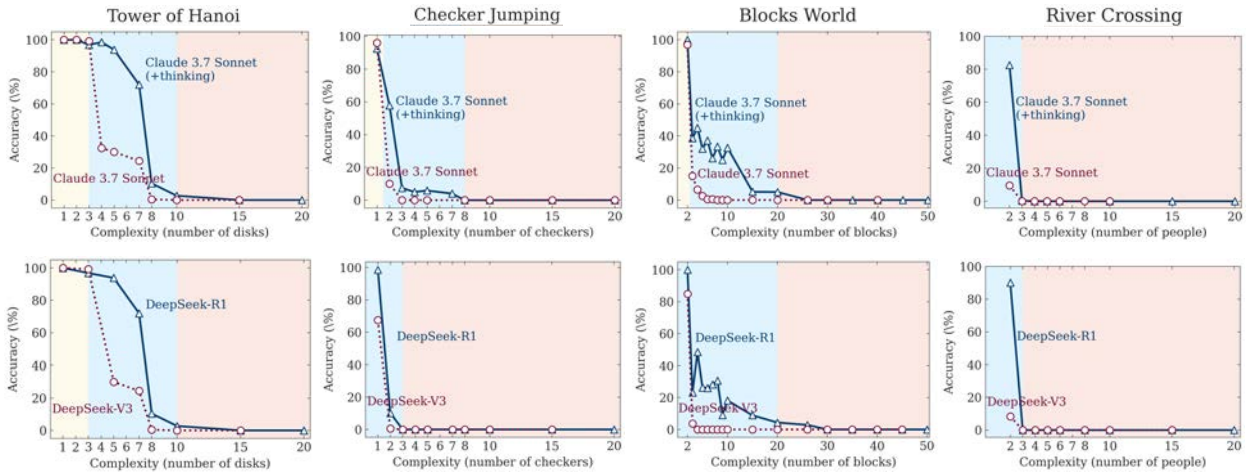


Figure 4: Accuracy of thinking models (Claude 3.7 Sonnet with thinking, DeepSeek-R1) versus their non-thinking counterparts (Claude 3.7 Sonnet, DeepSeek-V3) across all puzzle environments and varying levels of problem complexity.

4 Experiments & Results

4.1 Experimental Setup

Most of our experiments are conducted on reasoning models and their non-thinking counterparts, such as Claude 3.7 Sonnet (thinking/non-thinking) and DeepSeek-R1/V3. We chose these models because they allow access to the thinking tokens, unlike models such as OpenAI's o-series. For experiments focused solely on final accuracy, we also report results on the o-series models. For Claude 3.7 Sonnet models, we allow the maximum token budget (64k). Similarly, for DeepSeek-R1/V3 models on local servers, we allow the maximum length to be up to 64k tokens. For each puzzle instance, we generate 25 samples and report the average performance of each model across them. Comprehensive details of our experimental setup and results are provided in the Appendix.

4.2 How Does Complexity Affect Reasoning?

4.2.1 Three Regimes of Complexity

Motivated by the observations in Fig. 2, to systematically investigate the impact of problem complexity on reasoning behavior, we conducted experiments comparing **thinking** and **non-thinking** model pairs across our controlled puzzle environments. Our analysis focused on matched pairs of LLMs with identical model backbones, specifically *Claude-3.7-Sonnet (w. vs. w/o thinking)* and *DeepSeek (R1 vs. V3)*. In each puzzle, we vary the complexity by manipulating problem size N (representing disk count, checker count, block count, or crossing elements).

Fig. 4 presents the accuracy of both model types as a function of problem complexity across all puzzle environments. Complementing this, Fig. 5 shows the upper bound performance capabilities (pass@k) of these model pairs under equivalent inference token compute (averaged across all puzzles), extending earlier analyses from mathematical benchmarks (Fig. 2) to the controlled puzzle environments. Results from both these figures demonstrate that, unlike observations from math, there exists *three regimes* in the behavior of these models with respect to complexity. In the first regime where problem complexity is low, we observe that non-thinking models are capable to obtain performance comparable to, or even better than thinking models with more token-efficient inference. In the

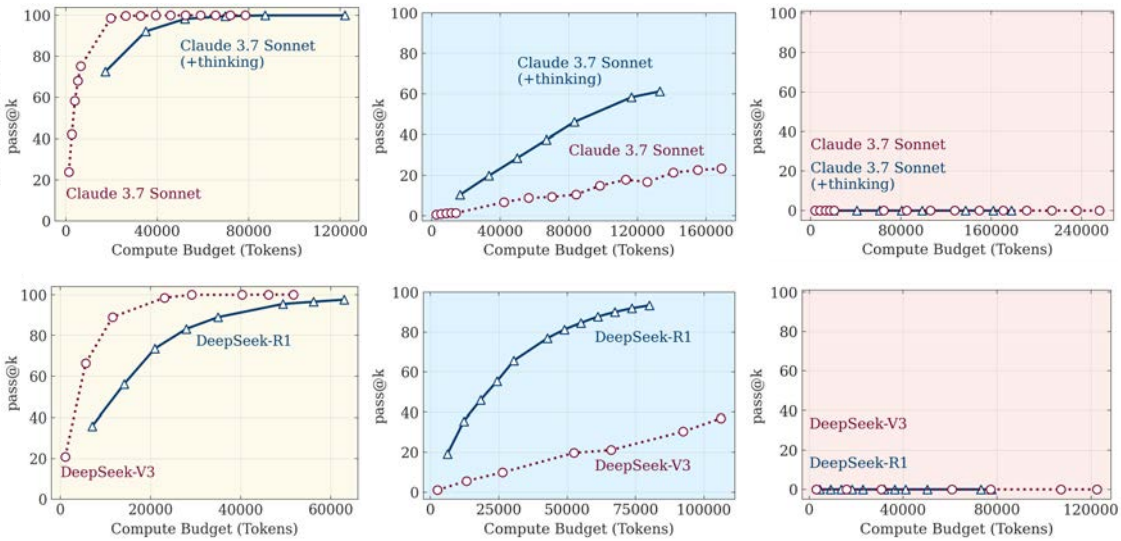


Figure 5: Pass@k performance of thinking vs. non-thinking models across equivalent compute budgets in puzzle environments of low, medium, and high complexity. Non-thinking models excel in simple problems, thinking models show advantages at medium complexity, while both approaches fail at high complexity regardless of compute allocation.

second regime with medium complexity, the advantage of reasoning models capable of generating long chain-of-thought begin to manifest, and the performance gap between model pairs increases. The most interesting regime is the third regime where problem complexity is higher and the performance of both models have collapsed to zero. Results show that while thinking models delay this collapse, they also ultimately encounter the same fundamental limitations as their non-thinking counterparts.

4.2.2 Collapse of Reasoning Models

We next examine how different specialized reasoning models equipped with thinking tokens respond to increasing problem complexity. Our experiments evaluate five state-of-the-art thinking models: *o3-mini* (medium and high configurations), *DeepSeek-R1*, *DeepSeek-R1-Qwen-32B*, and *Claude-3.7-Sonnet (thinking)*. Fig. 6 demonstrates these models' performance in terms of accuracy (top) and thinking token usage (bottom) across varying complexity levels. Results show that all reasoning models exhibit a similar pattern with respect to complexity: accuracy progressively declines as problem complexity increases until reaching complete collapse (zero accuracy) beyond a model-specific complexity threshold. Analysis of inference thinking token compute also reveals an intriguing pattern in thinking token allocation learned by these models. We observe that reasoning models initially increase their thinking tokens proportionally with problem complexity. However, upon approaching a critical threshold—which closely corresponds to their accuracy collapse point—models counterintuitively begin to reduce their reasoning effort despite increasing problem difficulty. This phenomenon is most pronounced in o3-mini variants and less severe in the Claude-3.7-Sonnet (thinking) model. Notably, despite operating well below their generation length limits with ample inference budget available, these models fail to take advantage of additional inference compute during the thinking phase as problems become more complex. This behavior suggests a fundamental scaling limitation in the thinking capabilities of current reasoning models relative to problem complexity.

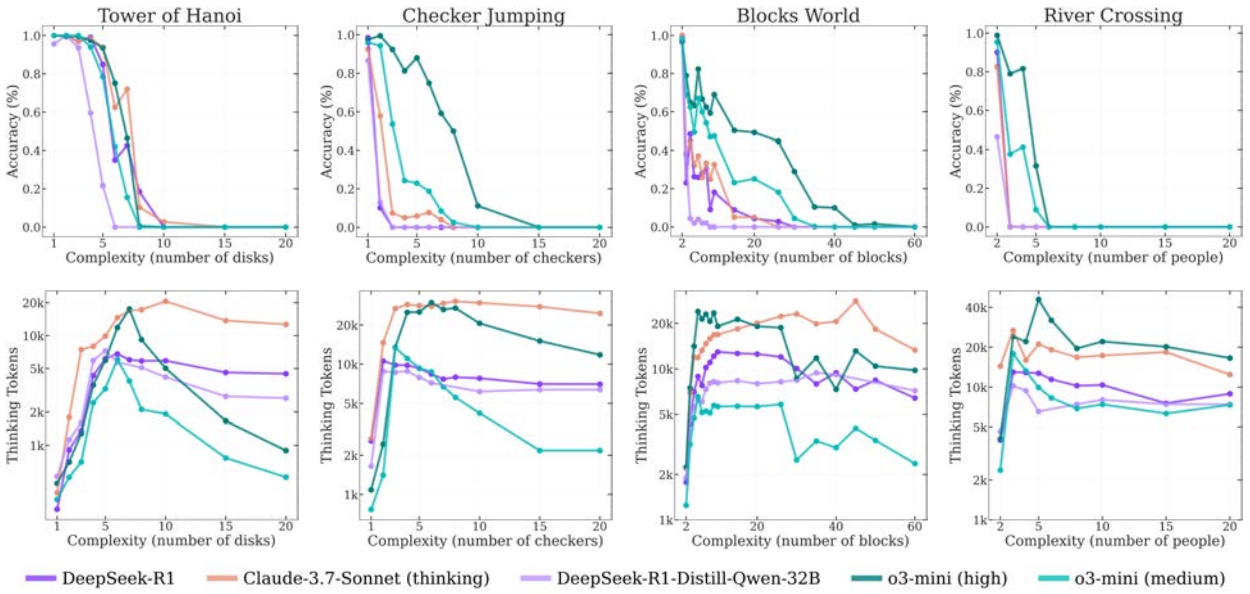
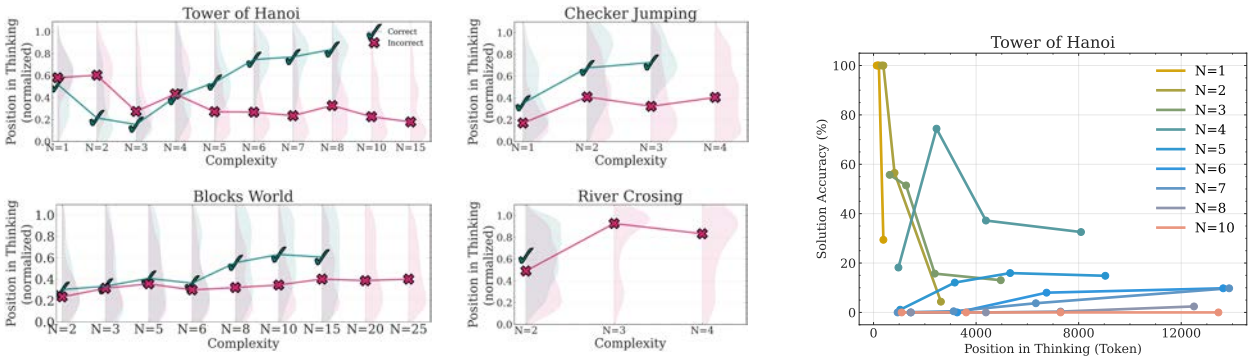


Figure 6: Accuracy and thinking tokens vs. problem complexity for reasoning models across puzzle environments. As complexity increases, reasoning models initially spend more tokens while accuracy declines gradually, until a critical point where reasoning collapses—performance drops sharply and reasoning effort decreases.

4.3 What Happens Inside the Thoughts of Reasoning Models?

To gain deeper insights into the thinking processes of reasoning models, we conducted a fine-grained analysis of their reasoning traces. As shown in Fig. 1, our setup with puzzle environments allows us to look beyond final answer and obtain more detailed insight into the reasoning traces ("thoughts") produced by these models. We extract and analyze the intermediate solutions explored *within the thoughts* of a model with the help of puzzle simulators. Our investigation examines the patterns and characteristics of these intermediate solutions, their correctness relative to their sequential position in the reasoning process, and how these patterns evolve with increasing problem complexity. For this analysis, we focus on the reasoning traces generated by *Claude-3.7-Sonnet-Thinking* across our puzzle suite. For each intermediate solution identified within the traces, we recorded: (1) its relative position within the reasoning trace (normalized by total thought length), (2) its correctness as validated by our puzzle simulators, and (3) the complexity of the corresponding problem. This allows to characterize the progression and accuracy of solution development throughout the reasoning process.

Fig. 7a demonstrates the relation between the position of intermediate solutions within thoughts, their correctness, and problem complexity across all puzzle environments. Our analysis from reasoning traces also further validates three regimes of complexity discussed above. For simpler problems, reasoning models often find the correct solution early in their thinking but then continue exploring incorrect solutions. Note the distribution of incorrect solutions (red) is shifted more upward towards end of thinking compared to correct solutions (green). This phenomenon, referred to as "overthinking" in the literature, leads to the waste of compute. As problems become moderately more complex, this trend reverses: models first explore incorrect solutions and mostly later in thought arrive at the correct ones. This time the distribution of incorrect solutions (red) is shifted more downward compared to correct ones (green). Finally, for the problems with higher complexity, collapse emerges,



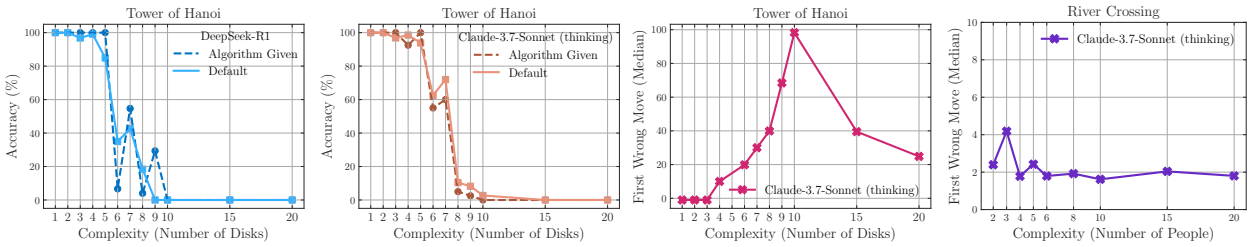
(a) (b)

Figure 7: **Left & Middle:** Position and correctness of intermediate solutions within reasoning traces across four puzzles at varying complexity levels. ✓ indicates correct solutions, ✗ indicates incorrect solutions, with distribution density shown by shading; **Right:** Solution accuracy versus position in thinking for Tower of Hanoi at different complexity levels. Simple problems (N=1-3) show early accuracy declining over time (overthinking), moderate problems (N=4-7) show slight improvement in accuracy with continued reasoning, and complex problems (N≥8) exhibit consistently near-zero accuracy, indicating complete reasoning failure.

meaning that the model fails to generate any correct solutions within the thought.
Fig. 7b presents a complementary analysis of solution accuracy within sequential segments (bins) of the thoughts in the Tower of Hanoi environment. It can be observed that for simpler problems (smaller N), solution accuracy tends to decrease or oscillate as thinking progresses, providing further evidence of the overthinking phenomenon. However, this trend changes for more complex problems, where solution accuracy increases with thinking progression—up to a certain threshold. Beyond this complexity threshold, in the "collapse mode", accuracy is zero.

4.4 Open Questions: Puzzling Behavior of Reasoning Models

In this section, we present surprising results concerning the limitations of reasoning models in executing exact problem-solving steps, as well as demonstrating different behaviors of the models based on the number of moves.
As shown in Figures 8a and 8b, in the Tower of Hanoi environment, even when we provide the algorithm in the prompt—so that the model only needs to execute the prescribed steps—performance does not improve, and the observed collapse still occurs at roughly the same point. This is noteworthy because finding and devising a solution should require substantially more computation (e.g., for search and verification) than merely executing a given algorithm. This further highlights the limitations of reasoning models in verification and in following logical steps to solve a problem, suggesting that further research is needed to understand the symbolic manipulation capabilities of such models [44, 6].
Moreover, in Figures 8c and 8d, we observe very different behavior from the Claude 3.7 Sonnet thinking model. In the Tower of Hanoi environment, the model's first error in the proposed solution often occurs much later, e.g., around move 100 for (N=10), compared to the River Crossing environment, where the model can only produce a valid solution until move 4. Note that this model also achieves near-perfect accuracy when solving the Tower of Hanoi with (N=5), which requires 31 moves, while it fails to solve the River Crossing puzzle when (N=3), which has a solution of 11 moves. This likely suggests that examples of River Crossing with N>2 are scarce on the web, meaning LRMs may not have frequently encountered or memorized such instances during training.



(a) (b) (c) (d)

Figure 8: **(a) & (b)** Despite providing the solution algorithm in the prompt, execution failure occurs at similar points, highlighting reasoning model limitations in logical step execution. **(c) & (d)** Notably, the Claude 3.7 Sonnet model demonstrates much longer error-free sequences in the Tower of Hanoi compared to early errors in the River Crossing scenario.

5 Conclusion

In this paper, we systematically examine frontier Large Reasoning Models (LRMs) through the lens of problem complexity using controllable puzzle environments. Our findings reveal fundamental limitations in current models: despite sophisticated self-reflection mechanisms, these models fail to develop generalizable reasoning capabilities beyond certain complexity thresholds. We identified three distinct reasoning regimes: standard LLMs outperform LRMs at low complexity, LRMs excel at moderate complexity, and both collapse at high complexity. Particularly concerning is the counterintuitive reduction in reasoning effort as problems approach critical complexity, suggesting an inherent compute scaling limit in LRMs. Our detailed analysis of reasoning traces further exposed complexity-dependent reasoning patterns, from inefficient "overthinking" on simpler problems to complete failure on complex ones. These insights challenge prevailing assumptions about LRM capabilities and suggest that current approaches may be encountering fundamental barriers to generalizable reasoning. Finally, we presented some surprising results on LRMs that lead to several open questions for future work. Most notably, we observed their limitations in performing exact computation; for example, when we provided the solution algorithm for the Tower of Hanoi to the models, their performance on this puzzle did not improve. Moreover, investigating the first failure move of the models revealed surprising behaviors. For instance, they could perform up to 100 correct moves in the Tower of Hanoi but fail to provide more than 5 correct moves in the River Crossing puzzle. We believe our results can pave the way for future investigations into the reasoning capabilities of these systems.

Limitations

We acknowledge that our work has limitations. While our puzzle environments enable controlled experimentation with fine-grained control over problem complexity, they represent a narrow slice of reasoning tasks and may not capture the diversity of real-world or knowledge-intensive reasoning problems. It is notable that most of our experiments rely on black-box API access to the closed frontier LRMs, limiting our ability to analyze internal states or architectural components. Furthermore, the use of deterministic puzzle simulators assumes that reasoning can be perfectly validated step by step. However, in less structured domains, such precise validation may not be feasible, limiting the transferability of this analysis to other more generalizable reasoning.

Acknowledgments

The authors would like to thank Scott Hoang, Yichen Jiang, Minsik Cho, Mohammad Sekhavat, David Harrison, Mohammadreza Armandpour and Devi Krishna for the valuable feedback and support.

References

[1] Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

[2] OpenAI. Introducing openai o1. Jan 2024.

[3] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[4] Anthropic. Claude 3.7 sonnet. Feb 2025.

[5] Google. Gemini flash thinking. *Google AI Blog*, Jan 2025.

[6] Seyed Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. GSM-symbolic: Understanding the limitations of mathematical reasoning in large language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

[7] Francois Chollet, Mike Knoop, Gregory Kamradt, Bryan Landers, and Henry Pinkard. Arc-agi-2: A new challenge for frontier ai reasoning systems. *arXiv preprint arXiv:2505.11831*, 2025.

[8] Gary Marcus. Five ways in which the last 3 months — and especially the deepseek era — have vindicated "deep learning is hitting a wall". Marcus on AI (Substack), February 2025. Blog post.

[9] Marah I Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat S. Behl, and et. al. Phi-3 technical report: A highly capable language model locally on your phone. *CoRR*, abs/2404.14219, 2024.

[10] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b. *CoRR*, abs/2310.06825, 2023.

[11] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurélien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Rozière, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny

Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Grégoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, and et al. The llama 3 herd of models. *CoRR*, abs/2407.21783, 2024.

[12] Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Sean Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Xiang Ren, Allyson Ettinger, Zaïd Harchaoui, and Yejin Choi. Faith and fate: Limits of transformers on compositionality. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

[13] R. Thomas McCoy, Shunyu Yao, Dan Friedman, Matthew Hardy, and Thomas L. Griffiths. Embers of autoregression: Understanding large language models through the problem they are trained to solve, 2023.

[14] Marianna Nezhurina, Lucia Cipolina-Kun, Mehdi Cherti, and Jenia Jitsev. Alice in wonderland: Simple tasks showing complete reasoning breakdown in state-of-the-art large language models. *arXiv preprint arXiv:2406.02061*, 2024.

[15] 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 Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[16] Mehran Kazemi, Najoung Kim, Deepti Bhatia, Xin Xu, and Deepak Ramachandran. Lambada: Backward chaining for automated reasoning in natural language. *arXiv preprint arXiv:2212.13894*, 2022.

[17] Hattie Zhou, Azade Nova, Hugo Larochelle, Aaron Courville, Behnam Neyshabur, and Hanie Sedghi. Teaching algorithmic reasoning via in-context learning. *arXiv preprint arXiv:2211.09066*, 2022.

[18] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213, 2022.

[19] Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Shengping Liu, Bin Sun, Kang Liu, and Jun Zhao. Large language models are better reasoners with self-verification. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 2550–2575, Singapore, December 2023. Association for Computational Linguistics.

[20] Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. Making language models better reasoners with step-aware verifier. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5315–5333, 2023.

[21] Eric Zhao, Pranjal Awasthi, and Sreenivas Gollapudi. Sample, scrutinize and scale: Effective inference-time search by scaling verification. *arXiv preprint arXiv:2502.01839*, 2025.

[22] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. STar: Bootstrapping reasoning with reasoning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[23] Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training language models with pause tokens. In *The Twelfth International Conference on Learning Representations*, 2024.

[24] David Herel and Tomas Mikolov. Thinking tokens for language modeling. *ArXiv*, abs/2405.08644, 2024.

[25] Zhihong Shao, Peiyi Wang, Runxin Xu Qihao Zhu, Junxiao Song, Mingchuan Zhang, Y.K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024.

[26] Amirhossein Kazemnejad, Milad Aghajohari, Eva Portelance, Alessandro Sordoni, Siva Reddy, Aaron Courville, and Nicolas Le Roux. Vineppo: Unlocking rl potential for llm reasoning through refined credit assignment, 2024.

[27] Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tülu 3: Pushing frontiers in open language model post-training. *ArXiv*, abs/2411.15124, 2024.

[28] Yanda Chen, Joe Benton, Ansh Radhakrishnan, Jonathan Uesato, Carson Denison, John Schulman, Arushi Somani, Peter Hase, Misha Wagner, Fabien Roger, et al. Reasoning models don't always say what they think. *arXiv preprint arXiv:2505.05410*, 2025.

[29] Dacheng Li, Shiyi Cao, Tyler Griggs, Shu Liu, Xiangxi Mo, Eric Tang, Sumanth Hegde, Kourosh Hakhamaneshi, Shishir G Patil, Matei Zaharia, et al. Llms can easily learn to reason from demonstrations structure, not content, is what matters! *arXiv preprint arXiv:2502.07374*, 2025.

[30] Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, et al. Do not think that much for 2+ 3=? on the overthinking of o1-like llms. *arXiv preprint arXiv:2412.21187*, 2024.

[31] Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Hanjie Chen, Xia Hu, et al. Stop overthinking: A survey on efficient reasoning for large language models. *arXiv preprint arXiv:2503.16419*, 2025.

[32] Sara Vera Marjanović, Arkil Patel, Vaibhav Adlakha, Milad Aghajohari, Parishad BehnamGhader, Mehar Bhatia, Aditi Khandelwal, Austin Kraft, Benno Krojer, Xing Han Lù, et al. Deepseek-r1 thoughtology: Let's< think> about llm reasoning. *arXiv preprint arXiv:2504.07128*, 2025.

[33] Yuxiao Qu, Matthew YR Yang, Amrith Setlur, Lewis Tunstall, Edward Emanuel Beeching, Ruslan Salakhutdinov, and Aviral Kumar. Optimizing test-time compute via meta reinforcement fine-tuning. *arXiv preprint arXiv:2503.07572*, 2025.

[34] Marthe Ballon, Andres Algaba, and Vincent Ginis. The relationship between reasoning and performance in large language models–o3 (mini) thinks harder, not longer. *arXiv preprint arXiv:2502.15631*, 2025.

[35] Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *arXiv preprint arXiv:2504.13837*, 2025.

[36] Benjamin Estermann, Luca A. Lanzendörfer, Yannick Niedermayr, and Roger Wattenhofer. Puzzles: A benchmark for neural algorithmic reasoning, 2024.

[37] Karthik Valmeekam, Alberto Olmo Hernandez, Sarath Sreedharan, and Subbarao Kambhampati. Large language models still can't plan (A benchmark for llms on planning and reasoning about change). *CoRR*, abs/2206.10498, 2022.

[38] Karthik Valmeekam, Kaya Stechly, and Subbarao Kambhampati. Llms still can't plan; can lrms? a preliminary evaluation of openai's o1 on planbench. 2024.

[39] Wenjie Ma, Jingxuan He, Charlie Snell, Tyler Griggs, Sewon Min, and Matei Zaharia. Reasoning models can be effective without thinking. *arXiv preprint arXiv:2504.09858*, 2025.

[40] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

[41] Mathematical Association of America. American invitational mathematics examination (aime). `https://maa.org/math-competitions/american-invitational-mathematics-examination-aime`, 2025. Accessed: 2025-05-15.

[42] Art of Problem Solving. Amc historical results - aime i (february 1, 2024). `https://artofproblemsolving.com/wiki/index.php/AMC_historical_results#AIME_I_.28February_1.2C_2024.29`, 2024. Accessed: 2025-05-15.

[43] Art of Problem Solving. Amc historical results – aime i (february 6, 2025). `https://artofproblemsolving.com/wiki/index.php/AMC_historical_results#AIME_I_.28February_6.2C_2025.29`, 2025. Accessed: 2025-05-15.

[44] Gary F Marcus. *The algebraic mind: Integrating connectionism and cognitive science*. MIT press, 2003.

[45] Saul Amarel. On representations of problems of reasoning about actions. In *Readings in artificial intelligence*, pages 2–22. Elsevier, 1981.

[46] Günter Rote. Crossing the bridge at night. *Bulletin of the EATCS*, 78:241, 2002.

A Appendix

In this appendix, we provide details supplementing the main text, including experimental setup specifications, additional results, and extended analysis.

A.1 Details on Puzzle Environment Specifications and Design - Comprehensive descriptions of all four puzzle environments, including their problem descriptions, prompt designs, and simulators.

- A.1.1 Tower of Hanoi
- A.1.2 Checker Jumping
- A.1.3 River Crossing
- A.1.4 Blocks World

A.2 Implementation Details - Full experimental setup specifications, model configurations, extraction pipeline details, and prescribed algorithm execution experiments.

A.3 Details on Computational Complexity

- A.3.1 Compositional Depth Characterization
- A.3.2 Performance vs Compositional Depth

A.4 Additional Results and Analysis - Extended analysis including reasoning effort patterns, and detailed failure analysis across all models and puzzle environments.

A.1 Details on Puzzle Environment Specifications and Design

A.1.1 Tower of Hanoi

Problem Description. The Tower of Hanoi is a classic recursive puzzle that serves as a great problem for evaluating sequential reasoning and planning capabilities in reasoning models. The puzzle consists of three pegs (labeled 0, 1, and 2 from left to right) and N disks of varying sizes, where each disk is uniquely numbered from 1 (smallest) to N (largest). In the initial configuration, all N disks are stacked on the leftmost peg (peg 0) in descending order of size, with the largest disk at the bottom and the smallest at the top. The remaining two pegs (1 and 2) are initially empty. The goal is to transfer all disks from peg 0 to peg 2, maintaining the same size ordering (largest at bottom, smallest at top). This puzzle is governed by three fundamental constraints: (1) *Single Disk Movement:* Only one disk may be moved at a time; (2) *Top Disk Access:* Only the topmost disk from any peg can be selected for movement; and (3) *Size Ordering Constraint:* A larger disk may never be placed on top of a smaller disk. This puzzle is a good evaluation testbed for reasoning and planning capabilities of models as it requires models to demonstrate key cognitive demands such as breaking down the problem into subproblems (recursive thinking), tracking multiple states and disk positions simultaneously (working memory management), adhering to movement rules and constraints while planning ahead (constraint satisfaction), and determining the correct order of operations to achieve the final goal (sequential planning).

The minimum number of moves required to solve the Tower of Hanoi recursive puzzle with N disks is $2^N - 1$, making it an exponentially scaling problem. This property allows for fine-grained difficulty control by adjusting the problem size with number of initial disks. However, in our evaluation framework, we focus on solution correctness rather than optimality, assessing each of the move's validity and the model's ability to reach the target state as the success criteria.

Prompt Design. The system prompt begins with a clear problem statement describing the puzzle setup. It explicitly states the movement rules and the objective of transferring all disks to the third peg. To facilitate understanding, the prompt includes example demonstrations as well as the critical formatting and reasoning expectations.

System Prompt - Tower of Hanoi

You are a helpful assistant. Solve this puzzle for me.
There are three pegs and n disks of different sizes stacked on the first peg. The disks are numbered from 1 (smallest) to n (largest). Disk moves in this puzzle should follow:

1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one stack and placing it on top of another stack.
3. A larger disk may not be placed on top of a smaller disk.

The goal is to move the entire stack to the third peg.
Example: With 3 disks numbered 1 (smallest), 2, and 3 (largest), the initial state is [[3, 2, 1], [], []], and a solution might be:

```
moves = [[1, 0, 2], [2, 0, 1], [1, 2, 1], [3, 0, 2],
         [1, 1, 0], [2, 1, 2], [1, 0, 2]]
```

This means: Move disk 1 from peg 0 to peg 2, then move disk 2 from peg 0 to peg 1, and so on.
Requirements:

- When exploring potential solutions in your thinking process, always include the corresponding complete list of moves.
- The positions are 0-indexed (the leftmost peg is 0).
- Ensure your final answer includes the complete list of moves in the format:
  `moves = [[disk id, from peg, to peg], ...]`

The user prompt after the system prompt presents the specific puzzle instance with current configuration showing the distribution of disks across pegs and the goal configuration specifying the target state.

User Prompt Template for N Disks - Tower of Hanoi

I have a puzzle with N disks of different sizes with
Initial configuration:

- Peg 0: N (bottom), ... 2, 1 (top)
- Peg 1: (empty)
- Peg 2: (empty)

Goal configuration:

- Peg 0: (empty)
- Peg 1: (empty)
- Peg 2: N (bottom), . . . 2, 1 (top)

Rules:

- Only one disk can be moved at a time.
- Only the top disk from any stack can be moved.
- A larger disk may not be placed on top of a smaller disk.

Find the sequence of moves to transform the initial configuration into the goal configuration.

Simulator. Our evaluation framework employs separate puzzle simulators for each puzzle to ensure rigorous and consistent assessment of solutions obtained from LRMs. The Tower of Hanoi simulator is designed as a stateful environment that tracks disk configurations across three pegs and validates each proposed move against the puzzle's fundamental constraints. The simulator architecture follows a modular design pattern with clear separation between state management, move validation, and solution verification. In this simulator, we have a puzzle class which tracks the current disk configuration and enforces the puzzle's fundamental constraints. We also have a method to execute each move in the puzzle setup and perform four-layer validation: checking peg boundary conditions (0-2), verifying source pegs contain disks, confirming the specified disk is topmost, and enforcing the size ordering constraint that prevents larger disks from being placed on smaller ones. Upon successful validation, the method executes the disk transfer and updates the game state. Then, the complete solution validation is processed by sequentially processing move lists, and verifying goal state achievement.

A.1.2 Checker Jumping

Problem Description. Checker Jumping is a one-dimensional constraint-satisfaction puzzle designed to test sequential reasoning, planning, and rule understanding capabilities. The puzzle consists of a linear arrangement of red checkers ('R'), blue checkers ('B'), and a single empty space ('_'). In the standard configuration, N red checkers are positioned on the left side, followed by an empty space in the middle, and N blue checkers on the right side, forming a linear board of length $2N + 1$. The objective is to swap the positions of all red and blue checkers, effectively mirroring the initial configuration, where red checkers end up on the right and blue checkers on the left. Movement in this puzzle is governed by two fundamental rules: (1) *Slide Movement:* A checker can slide forward into an adjacent empty space; and (2) *Jump Movement:* A checker can jump forward over exactly one checker of the opposite color to land in an empty space. Therefore, checkers cannot move backward toward their starting side—red checkers can only move rightward, and blue checkers can only move leftward from the initial configuration. This puzzle presents cognitive challenges that make it a great testbed for reasoning models. For example, models must demonstrate some aspect of spatial reasoning (tracking checker positions and possible moves), constraint satisfaction (adhering to movement rules during puzzle), lookahead planning (anticipating how current moves affect future

possibilities towards goal), and state-space exploration (searching through possible move sequences to find a valid solution path).
The difficulty of the Checker Jumping puzzle scales with the number of checkers: with N checkers of each color, the minimum solution requires $(N+1)^2 - 1$ moves, creating a quadratic relationship between problem size and solution complexity. In our evaluation framework, we mainly focus on solution correctness rather than optimality, evaluating each move against the puzzle constraints and confirming that the final state matches the goal configuration. This approach allows us to precisely identify reasoning failures and constraint violations that might occur during the solution process.

Prompt Design. The system prompt begins with a clear problem statement describing the puzzle setup and movement rules. It explicitly states the objective and provides a concrete example with a small board configuration to illustrate how moves should be represented.

System Prompt - Checker Jumping

You are a helpful assistant. Solve this puzzle for me.
On a one-dimensional board, there are red checkers ('R'), blue checkers ('B'), and one empty space ('_'). A checker can move by either:

1. Sliding forward into an adjacent empty space, or
2. Jumping over exactly one checker of the opposite color to land in an empty space.

The goal is to swap the positions of all red and blue checkers, effectively mirroring the initial state.
Example: If the initial state is ['R', '_', 'B'], the goal is to reach ['B', '_', 'R']. Your solution should be a list of moves where each move is represented as [checker_color, position_from, position_to]. For example:

```
moves = [['R', 0, 1], ['B', 2, 0], ['R', 1, 2]]
```

This means: Move the red checker from position 0 to 1, then move the blue checker from position 2 to 0, and so on.
Requirements:

- When exploring potential solutions in your thinking process, always include the corresponding complete list of moves.
- The positions are 0-indexed (the leftmost position is 0).
- Ensure your final answer includes the complete list of moves for final solution in the format: `moves = [[checker_color, position_from, position_to], ...]`

The user prompt presents the specific puzzle instance with the initial board configuration, and the goal state.

User Prompt Template for N Checkers - Checker Jumping

I have a puzzle with 2N+1 positions, where N red checkers ('R') on left, N blue checkers ('B') on right, and one empty space ('_') in between are arranged in a line.
Initial board: R R ... R _ B B ... B
Goal board: B B ... B _ R R ... R
Rules:

- A checker can slide into an adjacent empty space.
- A checker can jump over exactly one checker of the opposite color to land in an empty space.
- Checkers cannot move backwards (towards their starting side).

Find the minimum sequence of moves to transform the initial board into the goal board.

Simulator. Our evaluation framework employs a custom simulator for validating Checker Jumping puzzle solutions. The simulator implements a comprehensive validation system that enforces all puzzle constraints while tracking the state evolution throughout the solution path. The Checker Jumping simulator is designed as a stateful environment that tracks the position of all checkers and the empty space, validating each move of a given solution against the puzzle's movement rules. The simulator begins by validating that both the initial and goal states are well-formed, containing the same number of red and blue checkers and exactly one empty space. Then, each move is executed with a method that performs multi-layer validation: verifying position boundaries, confirming correct checker color at source, ensuring target positions are empty, and validating move types as either slides (distance=1) or jumps (distance=2). The simulator enforces directional constraints preventing backward movement (red checkers move right, blue checkers move left) and validates jump moves by confirming the presence of an opposite-colored checker in the middle position. Upon successful validation, the method executes the checker transfer by updating positions and clearing the source. Then, the complete move sequences are processed with final goal state verification.

A.1.3 River Crossing

Problem Description. River Crossing is a constraint satisfaction planning puzzle that tests multi-agent coordination and constraint management. This puzzle is a generalization of classic problems such as the Missionaries and Cannibals problem and the Bridge and Torch problem, which have been widely studied in planning literature [45, 46]. The river crossing puzzle involves N actors (denoted by a_1, a_2, ..., a_N) and their corresponding N agents (denoted by A_1, A_2, ..., A_N) who must cross a river using a boat. In the initial state, all $2N$ individuals are on the left bank of the river. The goal is to transport everyone safely to the right bank. The puzzle operates under several key movement constraints: (1) *Boat Capacity Constraint:* The boat can carry at most k individuals at a time, where k is typically set to 2 for smaller puzzles ($N \leq 3$) and 3 for larger puzzles ($N \leq 5$); (2) *Non-Empty Boat Constraint:* The boat cannot travel empty and must have at least one person aboard; (3) *Safety Constraint:* An actor cannot be in the presence of another agent unless their own agent is also present, as agents must protect their clients from competing agents. This safety constraint applies both on the banks and in the boat. This puzzle requires complex planning and state tracking as participants must carefully coordinate their crossings while maintaining safety constraints at all times. The solver must reason through

different combinations of individuals who can safely travel together, determine who should return with the boat after a crossing, and strategically plan a sequence that eventually brings everyone to the right bank without violating any constraints. The complexity of this task can be controlled by adjusting the number of actor-agent pairs and the boat capacity, creating a scalable challenge for reasoning models.

Prompt Design. The system prompt introduces the notation for representing actors and agents, establishes the solution format as a list of boat moves, and provides a simple example to demonstrate the format.

System Prompt - River Crossing

You are a helpful assistant. Solve this puzzle for me.
You can represent actors with a_1, a_2, ... and agents with A_1, A_2, ... Your solution must be a list of boat moves where each move indicates the people on the boat. For example, if there were two actors and two agents, you should return:

```
moves=[["A_2", "a_2"], ["A_2"], ["A_1", "A_2"], ["A_1"], ["A_1", "a_1
    "]]
```

which indicates that in the first move, A_2 and a_2 row from left to right, and in the second move, A_2 rows from right to left and so on.
Requirements:

- When exploring potential solutions in your thinking process, always include the corresponding complete list of boat moves.
- The list shouldn't have comments.
- Ensure your final answer also includes the complete list of moves for final solution.

The user prompt presents the specific puzzle instance with N actor-agent pairs, and the boat capacity k, and the safety constraint that must be maintained throughout the solution.

User Prompt Template for N Pairs - River Crossing

N actors and their N agents want to cross a river in a boat that is capable of holding only k people at a time, **with the constraint that no actor can be in the presence of another agent, including while riding the boat, unless their own agent is also present**, because each agent is worried their rivals will poach their client. Initially, all actors and agents are on the left side of the river with the boat. How should they cross the river? (Note: the boat cannot travel empty)

Simulator. Our evaluation framework employs a custom simulator for validating River Crossing puzzle extracted solutions. The simulator tracks the state of all individuals (actors and agents) and the boat position while enforcing all puzzle constraints. Each move is executed with multi-step validation: checking boat capacity limits, verifying all passengers are on the boat's current side, and enforcing the critical safety constraint that actors cannot be in the presence of other agents without their own agent present, both on the boat and on each bank after the move. The simulator

manages dynamic boat positioning, automatically switching sides after each crossing, and validates the complete state after each move to ensure no safety violations occur on either bank. Then, the complete crossing sequences are verified that all $2N$ individuals successfully reach the right bank.

A.1.4 Blocks World

Problem Description. Blocks World is a classical planning puzzle that has been recently studied for analyzing the planning capabilities of LLMs [37, 38]. The puzzle involves multiple stacks of blocks (A, B, C, etc.) that must be rearranged from an initial configuration to a specified goal configuration. Each block is uniquely identified by its letter, and the objective is to find the minimum sequence of moves needed to transform the initial state into the goal state. The puzzle operates only under two fundamental constraints: (1) *Top Block Movement:* Only the topmost block from any stack can be moved; and (2) *Valid Placement:* A block can only be placed either on an empty position or on top of another block. These constraints create planning problem where the order of operations becomes critical, as some configurations may require temporary placement of blocks to access those beneath them later. Blocks World serves as a great testbed for evaluating planning capabilities in reasoning models because it requires forward thinking, and state tracking. Recent studies have examined this puzzle in various configurations, including simplified settings with as few as 3 to 5 blocks, to evaluate LLM performance on sequential planning tasks [37, 38]. Models must demonstrate the ability to decompose complex state transformations into valid sequential moves, reason about dependencies between blocks (e.g., unblocking lower blocks before accessing them), and efficiently plan paths to the goal state without illegal moves.

The difficulty of this puzzle can be scaled by adjusting several parameters: the number of blocks, the number of stacks, and the complexity of the initial and goal configurations. We primarily control complexity through the block count N, while following clear structural patterns in the initial and goal configurations. In our experimental design, the initial configuration consistently divides the N blocks between two stacks in alphabetical order, with the third stack empty as workspace. The goal configuration consolidates all blocks onto the first stack in a systematic interleaved pattern that alternates between blocks from the two initial stacks, with specific positioning that requires complete disassembly and reassembly of the existing stacks. For example, for $N = 4$, the initial state has blocks divided between two stacks `[["A", "B"], ["C", "D"], []]` and the goal state `[["D", "B", "C", "A"], [], []]` requires interleaving blocks from both stacks; and for $N = 6$, the initial state `[["A", "B", "C"], ["D", "E", "F"], []]` must be transformed to `[["F", "C", "E", "B", "D", "A"], [], []]`, forming a complex alternating pattern. As N increases, the state space grows factorially, and the minimum solution length increases approximately linearly with N. For small values of N (2-7), the puzzles test basic planning; for medium values (8-20), they require more complex reasoning with longer planning horizons; and for large values ($N > 20$), they challenge the limits of sequential reasoning capabilities by requiring extensive temporary movements and pattern recognition across lengthy solution paths.

Prompt Design. The system prompt introduces the fundamental rules of the Blocks World puzzle, establishes the move representation format, and provides a simple example to demonstrate the solution structure.

System Prompt - Blocks World

You are a helpful assistant. Solve this puzzle for me.
In this puzzle, there are stacks of blocks, and the goal is to rearrange them into a target configuration using a sequence of moves where:

- Only the topmost block from any stack can be moved.
- A block can be placed either on an empty position or on top of another block.

Example: With initial state `[["A", "B"], ["C"], []]` and goal state `[["A"], ["B"], ["C"]]`, a solution might be:

```
moves = [["C", 1, 2], ["B", 0, 1]]
```

This means: Move block C from stack 1 to stack 2, then move block B from stack 0 to stack 1.
Requirements:

- When exploring potential solutions in your thinking process, always include the corresponding complete list of moves.
- Ensure your final answer also includes the complete list of moves for final solution in the format: `moves = [[block, from stack, to stack], ...]`

The user prompt presents the specific puzzle instance with the initial and goal configurations provided, and explicitly reminds the model about the movement constraint.

User Prompt Template for N Blocks - BlocksWorld

I have a puzzle with N blocks.
Initial state:

Stack 0: $blocks_0$ (top)

Stack 1: $blocks_1$ (top)

...

Stack m: $blocks_m$ (top)

Goal state:

Stack 0: $goal_blocks_0$ (top)

Stack 1: $goal_blocks_1$ (top)

...

Stack m: $goal_blocks_m$ (top)

Find the minimum sequence of moves to transform the initial state into the goal state. Remember that only the topmost block of each stack can be moved.

Simulator. Our evaluation framework employs a custom simulator for validating Blocks World puzzle extracted solutions. The simulator manages the state of all blocks across stacks while enforcing the puzzle's movement constraints. Each move is executed in the puzzle setup with three-layer validation: verifying stack indices are within bounds, confirming the source stack contains blocks, and ensuring the specified block is at the top of its stack (enforcing the top-block-only movement rule). Upon successful validation, the block transfer is executed and the block is popped from the source stack and appended to the destination stack. Finally, the complete solution sequences of block movements are processed and verified that the resulting configuration matches the target goal state.

A.2 Implementation Details

Configurations Our experiments primarily utilized reasoning models and their non-thinking counterparts to enable thorough analysis of the thinking process. We specifically selected Claude 3.7 Sonnet (thinking/non-thinking) and DeepSeek-R1/V3 due to their ability to provide access to thinking traces, a critical requirement for our analysis. For experiments focused solely on final accuracy metrics, we also included results from OpenAI's o3-mini models, as they lack access to thoughts. For Claude 3.7 Sonnet (thinking and non-thinking) models we used maximum generation budget of 64,000 tokens, accessed through the API interface. Temperature is set to 1.0 for all API rus (Claude-3.7-Sonnet and o3-mini runs). The experiments with DeepSeek-R1, DeepSeek-V3, and DeepSeek-R1-Distill-Qern-32B are conducted on local servers with maximum generation length set to 64,000 and temperature set to 1.0. In all experiments, we generated 25 samples per puzzle instance at each complexity level (N value) and reported performance averages across all samples.

Solution Extraction A custom extraction pipeline was developed to process model responses and intermediate reasoning traces (thoughts). The pipeline consists of several key components. We implemented a flexible regex-based extractors to identify potential solution attempts in both the final response and thinking trace. The extraction process identify solution patterns using regular expressions (both explicit "`moves =`" patterns and alternative bracket-based solutions). We process and clean each extracted candidate solution by (*i*) Removing comments from the list (text following "#" in any line), and (*ii*) Normalizing move formats to what suggested in context to ensure consistent structure. Then, we validate solution format and structure to filter out invalid matches. During the extraction, we also capture metadata of token position for each extracted solution. Notably, for accurate position tracking within thinking traces, we employed the same tokenizer (`cl100k_base`) as the corresponding model to count tokens across all experiments. Token positions were also normalized with respect to thought length to enable cross-sample comparison. Finally, we make sure that the recorded solutions within the thought trace are unique and duplicate solutions (identical moves list) were filtered. In case of duplicate solutions, only the first solution is recorded for analysis.

Solution Evaluation After extraction, each solution candidate is passed to the corresponding simulator of puzzle for fine-grained verification. The simulator takes a solution as list of moves and evaluate that with respect to the puzzle (check App. A.1 for details of each puzzle simulator). Each move in the compositional solution is executed sequentially according to previous moves and the puzzle rules. Then, the final state obtained from all moves in the sequence is compared to the goal state of puzzle to determine full solution correctness. For incorrect solutions, details of first failure move and the type of failure is also collected during the move verification with puzzle simulator.

Execution of Prescribed Steps In addition to open-ended problem solving across different puzzles, we also conducted focused experiments to test how providing the explicit solving algorithm

guidance with prescribed steps would affect behavior of these reasoning models (Sec. 4.4).
We expected that finding and devising solution from scratch should require substantially more computation for model (e.g., for search and verification) than just following a given algorithm's steps. However, results in Figures 8a and 8b show that reasoning models' behavior does not change that much and the collapse still occurs at roughly same points as before with this setting. This finding strengthens evidence that the limitation is not just in problem-solving and solution strategy discovery but also in consistent logical verification and step execution limitation throughout the generated reasoning chains.
For example, models are provided with a complete recursive algorithm of solving Tower of Hanoi puzzle as follows. This algorithm scratchpad was appended to the standard problem prompt to test its impact on reasoning behavior.

Example of Prescribed Algorithm for Tower of Hanoi

Here is a pseudocode of recursive algorithm to solve the puzzle:

```
ALGORITHM Solve(n, source, target, auxiliary, moves)
    // n = number of disks to move
    // source = starting peg (0, 1, or 2)
    // target = destination peg (0, 1, or 2)
    // auxiliary = the unused peg (0, 1, or 2)
    // moves = list to store the sequence of moves

    IF n equals 1 THEN
        // Get the top disk from source peg
        disk = the top disk on the source peg
        // Add the move to our list: [disk_id, source, target]
        ADD [disk, source, target] to moves
        RETURN
    END IF

    // Move n-1 disks from source to auxiliary peg
    Solve(n-1, source, auxiliary, target, moves)

    // Move the nth disk from source to target
    disk = the top disk on the source peg
    ADD [disk, source, target] to moves

    // Move n-1 disks from auxiliary to target
    Solve(n-1, auxiliary, target, source, moves)
END ALGORITHM
```

To solve the entire puzzle of moving n disks from peg 0 to peg 2:
1. Initialize an empty list 'moves'
2. Execute Solve(n, 0, 2, 1, moves)
3. The 'moves' list will contain the complete solution

Note: When executing this pseudocode, track which disk is currently on top of each peg. The disk IDs in the moves list should correspond to the actual disk being moved.
You can use this algorithm as a scratchpad to help you solve the problem step by step.

A.3 Details on Computational Complexity

A.3.1 Compositional Depth Characterization

Compositional depth is the number of sequential operations (i.e., moves) required to reach a full solution. Figure 9 demonstrates how this depth scales with problem size (N) across our four puzzle environments. Each puzzle has a distinct growth pattern, reflecting its underlying computational complexity. For example, Tower of Hanoi shows exponential growth ($2^N - 1$), and Checker Jumping displays quadratic scaling ($(N+1)^2 - 1$). The River Crossing and Blocks World puzzles show more moderate, near-linear growth with N. These varying compositional depth profiles enable us to evaluate how language reasoning models handle different types of sequential reasoning challenges and if their accuracy is always correlated with the compositional depth required to solve the puzzle. More details regarding this analysis is provided in Figure 10 in App. A.4.

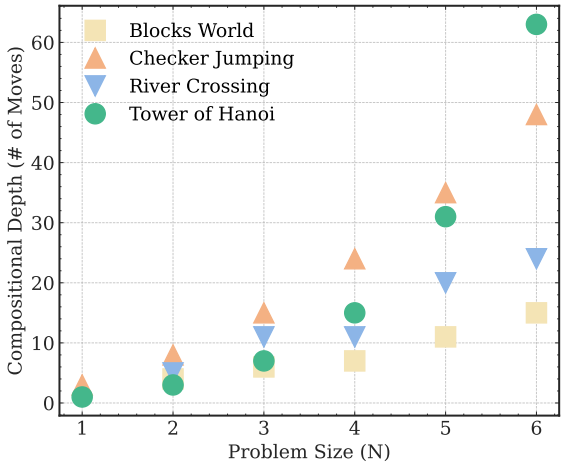


Figure 9: Compositional depth (number of moves required) across different problem sizes for our four puzzle environments.

A.3.2 Performance vs Compositional Depth

While intuition suggests a negative correlation between problem complexity and model accuracy, our analysis reveals a more nuanced relationship between compositional depth and LRM performance. Figure 10 demonstrates this across three state-of-the-art reasoning models (Claude-3.7-Sonnet w. thinking, DeepSeek-R1, and o3-mini) on our puzzle suite. Within individual puzzle types, we observe the expected negative correlation: as compositional depth increases, model accuracy consistently decreases. However, across different puzzle types, this relation breaks. Models may struggle with puzzles of lower compositional depth while succeeding on different puzzles with higher compositional depth. . For instance, models achieve >50% accuracy on Tower of Hanoi instances requiring approximately 10^2 moves, yet consistently fail on River Crossing puzzles with substantially lower compositional depth ($\sim 10^1$ moves).

A.4 Extended Results and Analysis

Failure Analysis. Understanding where models fail within the compositional reasoning steps provides insights beyond binary success metrics. Our accuracy evaluation requires perfect execution of entire move sequences—a single incorrect move results in failure. To examine failure patterns more granularly, we analyze the compositional depth at which models first make incorrect moves across varying problem complexity levels.

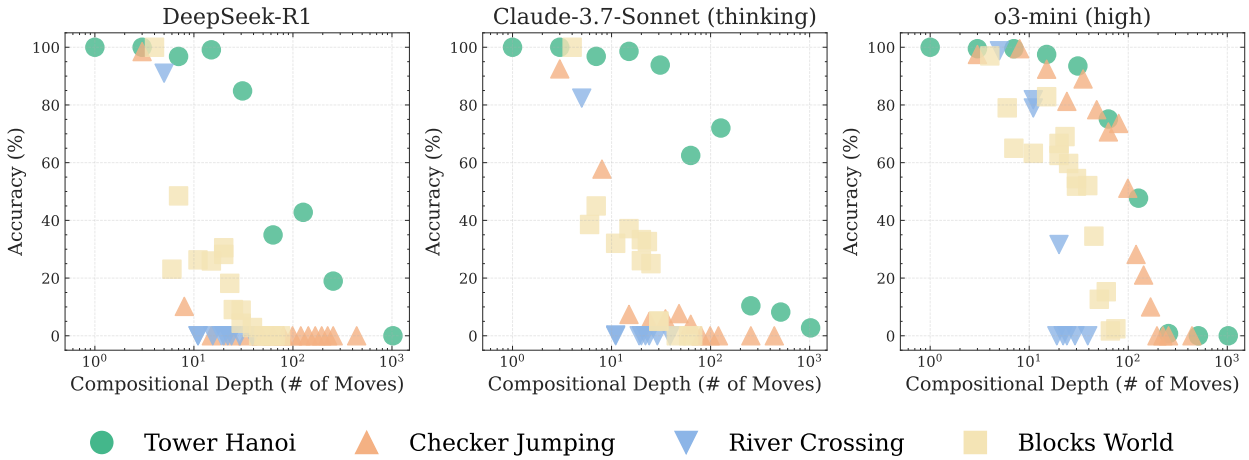


Figure 10: Accuracy versus compositional depth (number of moves required) for three LRMs (DeepSeek-R1, Claude-3.7-Sonnet with thinking, and o3-mini) across four puzzle environments.

Figure 11 shows the failure move ID versus problem complexity (N) within the solution sequence. The top row compares Claude-3.7-Sonnet with and without thinking capabilities, while the bottom row compares DeepSeek-R1 (thinking) with DeepSeek-V3 (non-thinking). These comparisons demonstrates how thinking mechanisms of LRMs influence failure patterns in compositional reasoning tasks of puzzles. Several counterintuitive patterns emerge from our analysis. First, models exhibit non-monotonic failure behavior with respect to problem complexity—instances where models fail earlier in the solution sequence for higher N values despite requiring longer overall solutions. For example, in Tower of Hanoi, models sometimes fail at below 50 moves for $N = 15$ but succeed through more than 100 moves for $N = 8$, contradicting the expectation that effective algorithmic planning and execution for the same puzzle should maintain consistent failure patterns relative to solution progress. This suggests fundamental inconsistencies in how models (both LRMs and their non-thinking standard LLM counterparts) apply learned solution strategies across different problem scales. Also, we observe that in the high-complexity regimes where both model variants experience complete accuracy collapse, e.g., Tower of Hanoi with $N \geq 15$ and Blocks World with $N \geq 40$, non-thinking models occasionally sustain performance deeper into the solution sequence and are able to fail at later moves than thinking-enabled variants. This is interesting as it shows that compositional reasoning failures in LLMs are not simply due to insufficient context length or inference compute, but rather reflect fundamental limitations in how models maintain algorithmic consistency across problem scales. We also analyze the distributional characteristics of failure moves to understand the consistency and reliability of model reasoning. Figure 12 presents the density distributions of failure move positions aggregated across all problem complexities for each puzzle environment, comparing thinking and non-thinking models within the same family. Based on the figure, thinking models (Claude-3.7-Sonnet with thinking and DeepSeek-R1) consistently show higher mean failure positions across all puzzles, as indicated by the dashed vertical lines showing mean of first failure in sequence of moves. However, the distribution shape of thinking models mostly have higher variance in their failure patterns. This suggests that while these models can reach deeper into solution sequences on average, their reasoning processes are more instable and prone to inconsistent performance.

Reasoning Effort Dynamics. Figure 13 demonstrates the reasoning effort (measured by inference thinking tokens) versus problem complexity across our puzzle environments. Green dots indicate

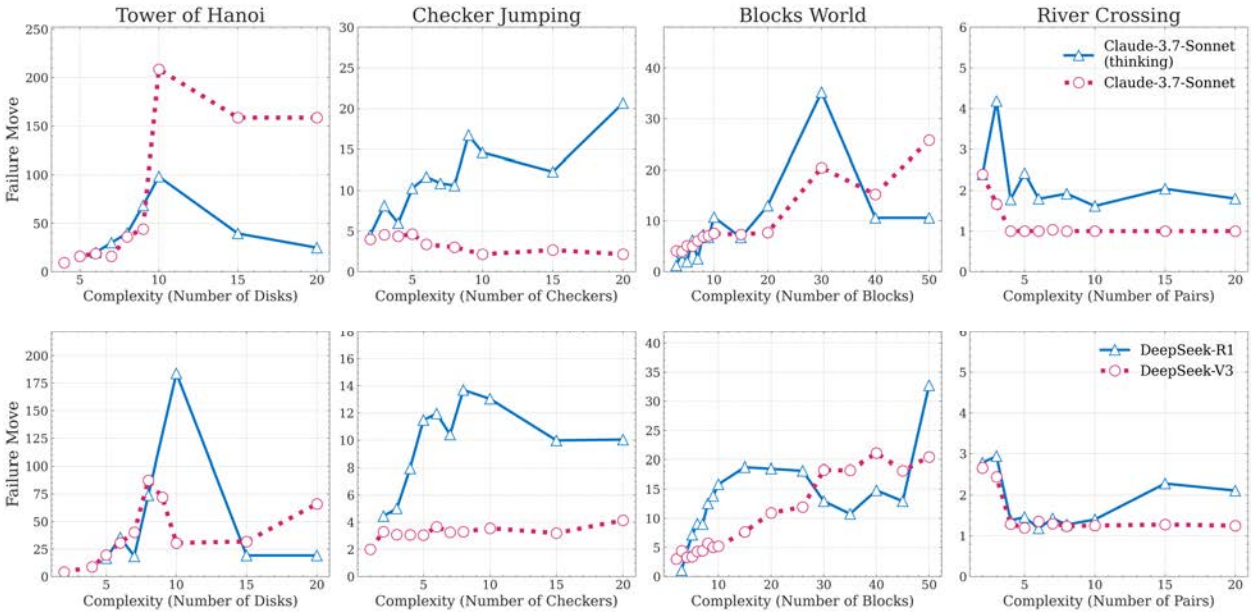


Figure 11: The first failure move versus problem complexity (N) comparison for thinking and non-thinking models across puzzle environments. **Top**: Claude-3.7-Sonnet comparison; **Bottom**: DeepSeek-R1 vs DeepSeek-V3.

correct solutions, red crosses show incorrect ones, and blue lines track average thinking token usage at each complexity level (N) across different puzzles and LRMs. We observe a consistent pattern across all three reasoning models (DeepSeek-R1, Claude-3.7-Sonnet-thinking, o3-mini) where thinking token usage, i.e. reasoning effort, initially scales with problem complexity but counterintuitively declines after reaching a model-specific threshold. This suggests an interesting and fundamental scaling limit in LRM thinking process for reasoning where beyond certain complexity thresholds, models not only fail to solve problems but counterintuitively reduce their inference compute despite facing more difficult problems and being well below the context and generation limits.

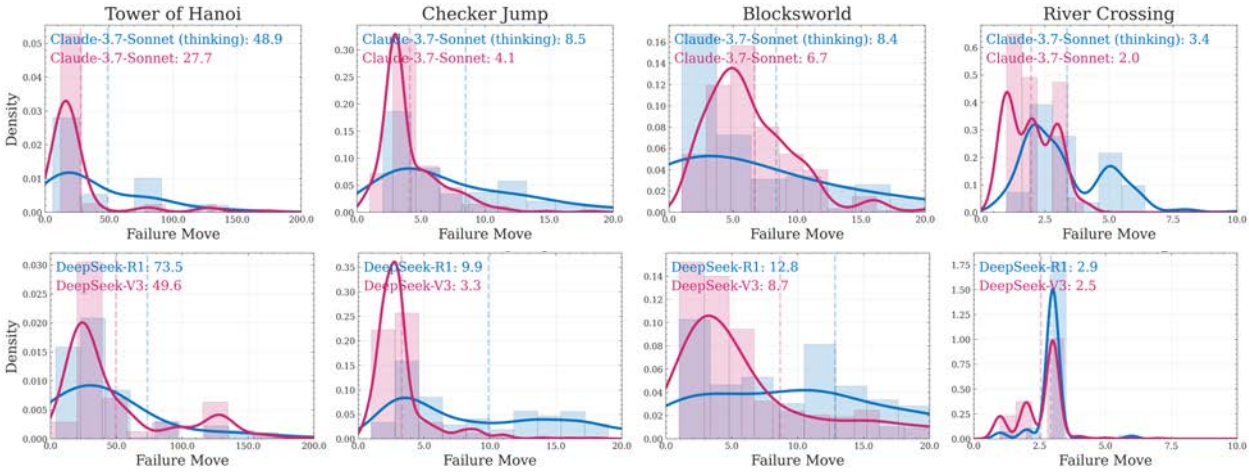


Figure 12: Density distribution of first failure moves for thinking and non-thinking models across puzzle environments. **Top**: Claude-3.7-Sonnet comparison; **Bottom**: DeepSeek-R1 vs DeepSeek-V3.

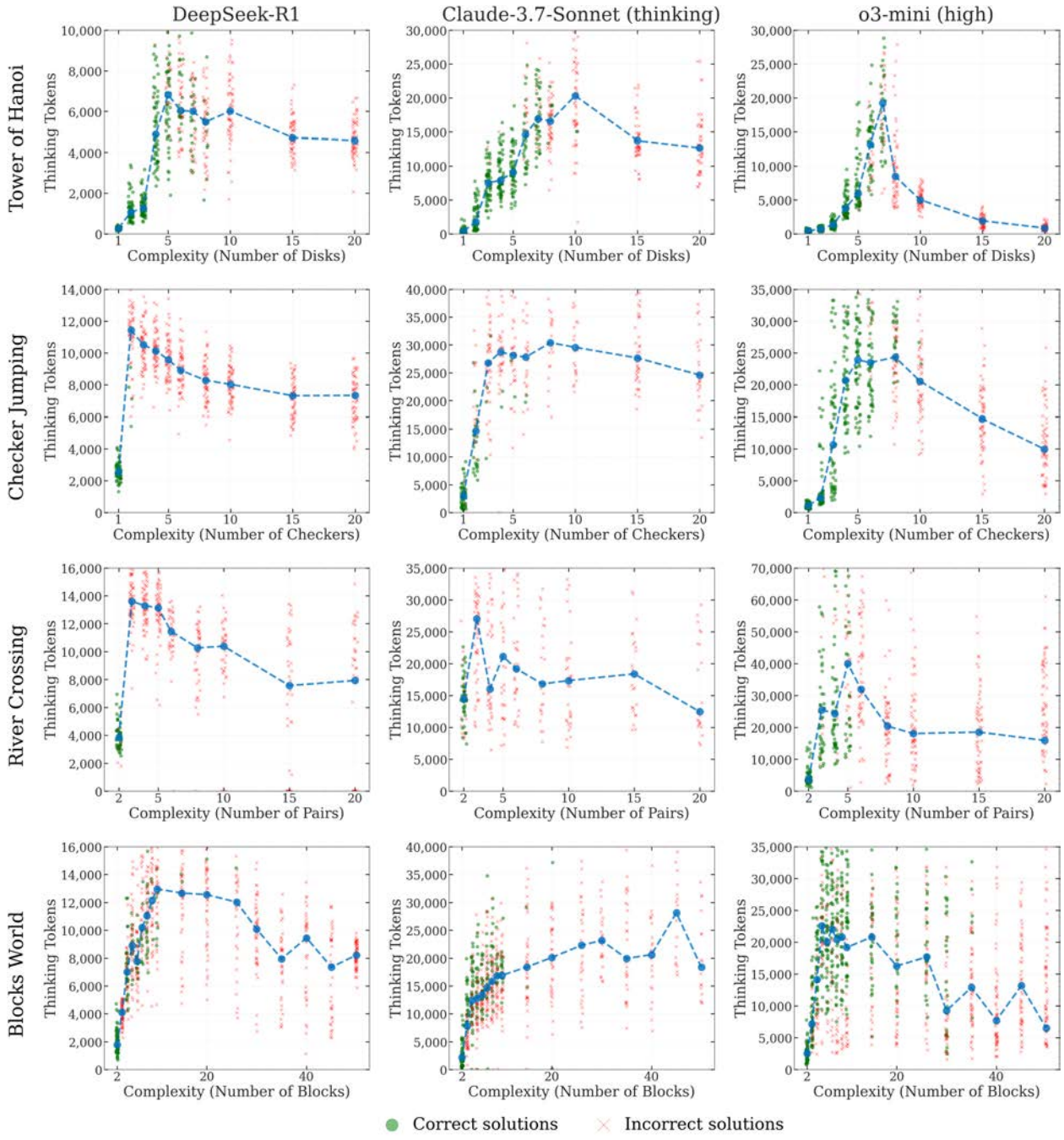


Figure 13: Detailed results on reasoning effort (measured in inference thinking tokens) versus problem complexity (N) for three LRMs (DeepSeek-R1, Claude-3.7-Sonnet with thinking, and o3-mini) across four puzzle environments.