Reproduction of Reasoning via Planning

Julius Šula
Institute of Computer Technology
TU Wien
Vienna, Austria
julius.sula@stud.tuwien.ac.at *

David Penz
Machine Learning Research Unit
TU Wien
Vienna, Austria

Abstract

This study examines the RAP (Reasoning via Planning) framework, focusing on its practical applicability in open-ended mathematical reasoning. What began as a reproduction study of RAP on the GSM8K benchmark revealed crucial insights about its computational requirements. While we successfully reproduce RAP's enhanced reasoning capabilities over Chain-of-Thought baselines, our detailed analysis of computational costs shows that RAP consistently requires an order of magnitude more tokens per iteration. This finding challenges the original paper's positioning of RAP for computationally constrained scenarios and suggests that its true strength emerges in high-compute environments where performance can be prioritized over efficiency. These insights provide important practical considerations for deploying advanced reasoning frameworks with LLMs and demonstrate how reproduction studies can reveal critical aspects of system behavior not apparent in initial publications.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable reasoning capabilities through Chain-of-Thought (CoT) prompting [9], enabling complex problem decomposition through intermediate steps. While these capabilities continue to advance, the reproducibility and practical implications of more sophisticated reasoning frameworks remain crucial research questions. The RAP framework [3] represents a significant advance in this direction by repurposing LLMs as both world model and reasoning agent, incorporating Monte Carlo Tree Search for systematic exploration of alternative reasoning paths.

Our study began as a reproduction effort of RAP's mathematical reasoning capabilities on the GSM8K benchmark. However, during our initial experiments with limited computational resources, we observed an unexpected pattern: while successfully reproducing RAP's reported accuracy improvements, we found its token consumption to be dramatically higher than baseline approaches. This observation led us to conduct a systematic investigation of RAP's computational efficiency—a critical consideration for scaling capabilities of LLMs given computational constraints.

In this work, we focus on two key questions:

- Can we validate RAP's reported performance advantages when comparing at equal iteration counts?
- How does RAP's true computational efficiency compare to CoT when normalizing for token consumption?

Our analysis reveals a striking contrast with the original paper's positioning: while successfully reproducing RAP's iteration-based improvements, we find that its token consumption is an order of

*Code available at: https://github.com/piragi/rap

magnitude higher than CoT approaches. This suggests that RAP's optimal use case lies not in computationally constrained scenarios as originally proposed, but rather in high-compute environments where breaking through performance plateaus takes precedence over efficiency.

2 Background

Large Language Models have demonstrated remarkable capabilities in few-shot learning settings [1, 5], with performance scaling predictably with model size and compute resources [4]. While scaling pre-training requires substantial computational resources and data, scaling inference-time computation offers a more accessible path to improving model capabilities.

Chain-of-Thought (CoT) prompting [9] represented a significant advance in LLM reasoning, enabling models to break down complex problems into intermediate steps. This approach has been further enhanced through techniques like self-consistency [8], which generates multiple reasoning paths and uses majority voting to select the final answer. However, these methods still rely on single-pass generations that limit systematic exploration of reasoning alternatives.

The success of Monte Carlo Tree Search (MCTS) in domains like AlphaGo [7] and MuZero [6] demonstrated how systematic exploration could enhance decision-making capabilities. Building on this insight, Tree of Thoughts [10] first applied search algorithms to LLM reasoning. The RAP framework [3] extended this approach by combining MCTS with a world model, where the LLM serves both as reasoning agent and world model. Recent work on speculative decoding [2] has begun addressing computational efficiency concerns, though detailed empirical analysis of token generation costs remains limited.

3 Method

The RAP framework formulates mathematical reasoning as a Markov Decision Process (MDP), where an LLM serves both as the reasoning agent and as a world model. For solving GSM8K math problems, this approach enables the systematic decomposition of complex questions into sub-questions while maintaining explicit tracking of intermediate calculations. The framework leverages Monte Carlo Tree Search (MCTS) to explore different reasoning paths, guided by a reward mechanism that evaluates both the quality of sub-questions and the consistency of intermediate answers.

3.1 World State Model

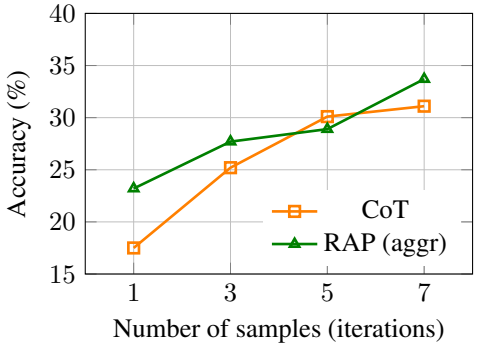
Mathematical reasoning in RAP is formulated as an MDP, where states represent our current knowledge at each reasoning step, while actions are questions that drive the reasoning forward. Consider a typical GSM8K problem: "A robe takes 2 bolts of blue fiber and half that much white fiber. How many bolts in total does it take?"

For such mathematical word problems:

- States (s_t) represent our accumulated knowledge at each step (e.g., "The robe needs 2 bolts of blue fiber")
- Actions (a_t) are formulated as sub-questions that drive the reasoning forward (e.g., "How much white fiber does the robe need?")

Starting from an initial state s_0, the LLM acting as a reasoning agent generates potential actions by sampling from a distribution $p(a|s_t, c)$, where c contains in-context demonstrations (see Appendix B for prompt details). Once an action a_t is selected, the same LLM, now serving as a world model, predicts the next state s_{t+1} through a state transition distribution $p(s_{t+1}|s_t, a_t, c')$, where c' is another prompt guiding the state generation. This process results in a reasoning trace consisting of interleaved states and actions $(s_0, a_0, s_1, ..., a_{t-1}, s_t)$.

Unlike previous methods such as Chain-of-Thought that only maintain a sequence of actions, this approach provides a more grounded reasoning process by explicitly tracking the world state at each step. The entire reasoning trace is simulated by the LLM without external interaction, similar to how humans might mentally explore different solution paths.



Method	Metric	Number of Iterations			
		1	3	5	7
CoT	Accuracy (%)	17.5	25.2	**30.1**	31.1
	Tokens	*79*	*228*	*421*	*560*
RAP	Accuracy (%)	**23.5**	**27.7**	28.9	**33.7**
	Tokens	*1,583*	*3,288*	*5,195*	*7,046*

Note: Token counts show average tokens generated per problem.

Figure 1: Comparison of RAP and Chain-of-Thought (CoT) performance on GSM8K. RAP shows consistent performance advantages over CoT when comparing at equal iteration counts, achieving 23.5% vs 17.5% accuracy at first iteration and 33.7% vs 31.1% at iteration 7, though with significantly higher token costs (see table).

3.2 Reward Mechanisms

The RAP framework employs two types of rewards that guide the search process. The first is a self-evaluation reward for actions, where the LLM assesses the quality of a proposed reasoning step. This lightweight reward can be quickly computed for candidate actions without generating their corresponding states, allowing for efficient action selection during search.

The second reward type evaluates states through a consistency mechanism. For a given state (answer to a sub-question), the LLM generates multiple samples and computes a confidence score based on the answer frequency. The final reward combines both the action's self-evaluation score and the state's consistency score through a weighted geometric mean $R = \text{self\_eval}^{\alpha} \cdot \text{consistency}^{(1-\alpha)}$, where α balances the influence of both components. This dual reward structure enables efficient exploration by allowing preliminary action evaluation without the computational cost of state generation, while still maintaining thorough evaluation of promising paths.

3.3 Monte Carlo Tree Search Implementation

Building on the world model and reward mechanisms, I implement a Monte Carlo Tree Search with Upper Confidence Bounds (UCT) for tree traversal. Each node in the search tree maintains its state, action, visit count, and cumulative reward. The search process follows the standard MCTS structure but adapts it for the language model reasoning domain.

During selection, nodes are chosen according to the UCT formula:

$$a^* = _{a \in A(s)} \left(Q(s,a) + W_{\text{EXP}} \sqrt{\frac{\ln N(s)}{N(s,a)}} \right)$$

where $Q(s,a)$ represents the node's value, $N(s)$ the parent's visit count, and $N(s,a)$ the node's visit count. The exploration weight W_{EXP} balances between exploring new paths and exploiting known high-reward sequences.

After selecting a leaf node, the expansion phase generates potential actions using the lightweight self-evaluation reward for efficiency. Simulation then performs rollouts to terminal states, and backpropagation updates the rewards along the selected path. Terminal states are reached either when a complete answer is generated or when hitting the maximum depth limit.

To leverage information across multiple search iterations, the final answers from terminal states are aggregated and weighted according to their cumulative rewards and tree depth. The confidence in the final prediction is computed as the proportion of total reward allocated to the selected answer, providing a measure of solution reliability.

4 Results

Our reproduction study used Llama 3.2 3B on a subset of 1000 samples from GSM8K, following the hyperparameter configuration of the original paper [3] with minor adjustments (see Appendix C for detailed setup). We evaluate RAP along two dimensions: solution accuracy and computational efficiency across multiple iterations. For accuracy, we measure the percentage of correctly solved problems, tracking performance improvements as iterations increase (from 1 to 7). For computational efficiency, we analyze the cumulative token generation overhead per iteration, enabling direct comparison with CoT baselines.

As shown in Figure 1, RAP demonstrates stronger initial performance compared to Chain-of-Thought (CoT) baselines, achieving 23.5% accuracy with a single iteration compared to CoT's 17.5% (see Appendix D for detailed statistical analysis). This performance advantage persists across iteration counts, culminating in 33.7% vs 31.1% accuracy at iteration 7. However, this comes at a dramatic difference in computational cost: RAP requires 1,583 tokens for a single iteration compared to CoT's 79 tokens—a 20x increase in token consumption.

Our analysis reveals that this computational overhead stems from fundamental differences in how the two methods scale. For Chain-of-Thought, the computational complexity grows linearly with iterations $O(n)$, as each iteration generates a single reasoning chain. In contrast, RAP's complexity grows as $O(n \cdot (b \cdot d))$, where b represents the branching factor at each node and d is the search tree depth. This manifests in our empirical measurements: at 5 iterations, RAP consumes 5,195 tokens compared to CoT's 421 tokens while achieving comparable accuracy (28.9% vs 30.1%).

When examining performance trends in our experiments, both methods show diminishing returns with increased computation. Moreover, prior work has demonstrated that CoT with self-consistency reaches a performance ceiling [8], while RAP has been shown to achieve higher maximum performance [3]. This pattern suggests that RAP's true advantage emerges specifically in high-compute scenarios where breaking through CoT's performance ceiling justifies the additional token overhead.

This insight fundamentally challenges the original paper's positioning of RAP for token-constrained scenarios. At equivalent token budgets, running multiple CoT iterations yields better performance than RAP's more complex exploration strategy for GSM8K-style problems. Recent work [2] reinforces these efficiency concerns, showing that even with optimizations like speculative decoding, MCTS-based approaches maintain similar token overhead ratios. However, in scenarios where maximum achievable performance takes precedence over computational efficiency, RAP's ability to surpass CoT's performance ceiling makes it the preferred choice despite its higher computational demands.

5 Conclusion

This work began as a reproduction study of the RAP framework but evolved into a broader investigation of its practical applicability. Our reproduction efforts strongly validate RAP's fundamental effectiveness, consistently outperforming CoT at equal iteration counts. However, our comprehensive token efficiency analysis reveals a crucial insight missing from the original work: while RAP achieves superior accuracy per iteration, it requires 12–20x more tokens to do so.

The key finding emerges from combining our computational analysis with insights from prior work on performance plateaus. Previous studies have shown that CoT with self-consistency reaches a performance ceiling that limits its maximum achievable accuracy [8], while RAP has demonstrated the potential to push beyond this ceiling—albeit at substantial computational cost. This observation fundamentally reframes RAP's optimal deployment strategy: rather than excelling in computationally constrained scenarios as originally proposed, its true strength emerges in high-compute contexts where breaking through performance plateaus takes precedence over computational efficiency.

While our study has limitations (see Appendix E), this insight about RAP's optimal use case—prioritizing maximum achievable performance over computational efficiency—provides crucial guidance for organizations balancing performance requirements against computational costs in advanced reasoning systems.

References

[1] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. URL http://arxiv.org/abs/2005.14165.

[2] Zitian Gao, Boye Niu, Xuzheng He, Haotian Xu, Hongzhang Liu, Aiwei Liu, Xuming Hu, and Lijie Wen. Interpretable Contrastive Monte Carlo Tree Search Reasoning. URL http://arxiv.org/abs/2410.01707.

[3] 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. URL http://arxiv.org/abs/2305.14992.

[4] 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. URL http://arxiv.org/abs/2001.08361.

[5] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie

Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, C. J. Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. GPT-4 Technical Report. URL http://arxiv.org/abs/2303.08774.

[6] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. 588(7839):604–609. ISSN 0028-0836, 1476-4687. doi: 10.1038/s41586-020-03051-4. URL http://arxiv.org/abs/1911.08265.

[7] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. 529(7587):484–489. ISSN 0028-0836, 1476-4687. doi: 10.1038/nature16961. URL https://www.nature.com/articles/nature16961.

[8] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-Consistency Improves Chain of Thought Reasoning in Language Models. URL http://arxiv.org/abs/2203.11171.

[9] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. URL http://arxiv.org/abs/2201.11903.

[10] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. URL http://arxiv.org/abs/2305.10601.

A Prompt Examples and Format

Our implementation follows the prompt design from the original RAP paper [3], with minor modifications based on early experiments with our chosen LLM. The core prompt structure supports both question generation and answer prediction, using the same format with different expected endings (questions vs. answers with explicit calculations).

During the tree search, we append the new problem as "Question 5" to the prompt examples. For each subsequent level in the search tree, all previously generated questions and their answers from the current reasoning path are also appended to the prompt. This ensures that the LLM maintains the full context of the current reasoning path when generating the next question or answer.

For detailed discussion of the prompt design principles and variations, we refer readers to the original paper [3].

Listing 1: Question Decomposition Prompt

```
Given a question, please decompose it into sub-questions. For each sub-question, please
    answer it in a complete sentence, ending with "The answer is". When the original
    question is answerable, please start the subquestion with "Now we can answer the
    question: ".
Question 1: Four years ago, Kody was only half as old as Mohamed. If Mohamed is
    currently twice as 30 years old, how old is Kody?
Question 1.1: How old is Mohamed?
Answer 1.1: He is currently 30 * 2 = 60 years old. The answer is 60.
Question 1.2: How old was Mohamed four years ago?
Answer 1.2: Four years ago, he must have been 60 - 4 = 56 years old. The answer is 56.
Question 1.3: How old was Kody four years ago?
Answer 1.3: Kody was half as old as Mohamed four years ago. Thus, Kody was 56 / 2 = 28
    years old. The answer is 28.
Question 1.4: Now we can answer the question: How old is Kody?
Answer 1.4: She is currently 28 + 4 = 32 years old. The answer is 32.

Given a question, please decompose it into sub-questions. For each sub-question, please
    answer it in a complete sentence, ending with "The answer is". When the original
    question is answerable, please start the subquestion with "Now we can answer the
    question: ".
Question 2: On a moonless night, three fireflies danced in the evening breeze. They
    were joined by four less than a dozen more fireflies before two of the fireflies
    flew away. How many fireflies remained?
Question 2.1: How many fireflies joined?
Answer 2.1: The fireflies were joined by four less than a dozen more fireflies, which
    are 12 - 4 = 8 fireflies. The answer is 8.
Question 2.2: Now we can answer the question: How many fireflies remained?
Answer 2.2: Three fireflies were dancing originally. They were joined by 8 fireflies
    before two of them flew away. So there were 3 + 8 - 2 = 9 remaining. The answer is
    9.

Given a question, please decompose it into sub-questions. For each sub-question, please
    answer it in a complete sentence, ending with "The answer is". When the original
    question is answerable, please start the subquestion with "Now we can answer the
    question: ".
Question 3: Ali has four $10 bills and six $20 bills that he saved after working for
    Mr. James on his farm. Ali gives her sister half of the total money he has and
    uses 3/5 of the remaining amount of money to buy dinner. Calculate the amount of
    money he has after buying the dinner.
Question 3.1: How much money does Ali have in total?
Answer 3.1: Ali has four $10 bills and six $20 bills. So he has 4 * 10 + 6 * 20 = 160
    dollars. The answer is 160.
Question 3.2: How much money does Ali give to his sister?
Answer 3.2: Ali gives half of the total money he has to his sister. So he gives 160 / 2
    = 80 dollars to his sister. The answer is 80.
Question 3.3: How much money does Ali have after giving his sister the money?
Answer 3.3: After giving his sister the money, Ali has 160 - 80 = 80 dollars left. The
    answer is 80.
```

```
Question 3.4: How much money does Ali use to buy dinner?
Answer 3.4: Ali uses 3/5 of the remaining amount of money to buy dinner. So he uses 80
    * 3/5 = 48 dollars to buy dinner. The answer is 48.
Question 3.5: Now we can answer the question: How much money does Ali have after buying
    the dinner?
Answer 3.5: After buying the dinner, Ali has 80 - 48 = 32 dollars left. The answer is
    32.

Given a question, please decompose it into sub-questions. For each sub-question, please
    answer it in a complete sentence, ending with "The answer is". When the original
    question is answerable, please start the subquestion with "Now we can answer the
    question: ".
Question 4: A car is driving through a tunnel with many turns. After a while, the car
    must travel through a ring that requires a total of 4 right-hand turns. After the
    1st turn, it travels 5 meters. After the 2nd turn, it travels 8 meters. After the
    3rd turn, it travels a little further and at the 4th turn, it immediately exits
    the tunnel. If the car has driven a total of 23 meters around the ring, how far
    did it have to travel after the 3rd turn?
Question 4.1: How far did the car travel except for the 3rd turn?
Answer 4.1: It travels 5 meters after the 1st, 8 meters after the 2nd, and 0 meters
    after the 4th turn. Its a total of 5 + 8 + 0 = 13 meters. The answer is 13.
Question 4.2: Now we can answer the question: How far did the car have to travel after
    the 3rd turn?
Answer 4.2: The car has driven a total of 23 meters around the ring. It travels 13
    meters except for the 3rd turn. So it has to travel 23 - 13 = 10 meters after the
    3rd turn. The answer is 10.

Given a question, please decompose it into sub-questions. For each sub-question, please
    answer it in a complete sentence, ending with "The answer is". When the original
    question is answerable, please start the subquestion with "Now we can answer the
    question: ".
Question 5: [current question]
```

For self-evaluation of reasoning steps, we use:

Listing 2: Self-evaluation reward prompt

```
Given a question and some sub-questions, determine whether the last sub-question is
    useful to answer the question. Output 'Yes' or 'No', and a reason.

Question 1: Four years ago, Kody was only half as old as Mohamed. If Mohamed is
    currently twice as 30 years old, how old is Kody?
Question 1.1: How old is Mohamed?
Question 1.2: How old was Mohamed four years ago?
New question 1.3: How old was Kody four years ago?
Is the new question useful? Yes. We need the answer to calculate how old is Kody now.

Question 2: Traci and Harris are baking cakes together. Traci has brought flour from
    her own house and Harris has 400g of flour in his house. Each cake needs 100g of
    flour and Traci and Harris have created 9 cakes each. How much flour, in grams,
    did Traci bring from her own house?
New question 2.1: How many cakes did Traci bring from her own house?
Is the new question useful? No. The new question is not related to the original
    question.

Question 3: A quantity surveyor is figuring the construction costs for a couple that
    wishes to build a house. The costs are as follows: land costs $50 per square
    meter, bricks cost $100 per 1000 bricks and roof tiles cost $10 per roof tile. If
    the house they wish to build requires 2000 square meters, 10000 bricks, and 500
    roof tiles, how much construction costs are required for this project?
Question 3.1: How much does the land cost?
Question 3.2: How much do the bricks cost?
New question 3.3: How much do the roof tiles cost?
Is the new question useful? Yes. We need the answer to calculate the total construction
    costs.
```

```
Question 4: Wallace's water heater is twice the size of Catherine's water heater. If
    the capacity of Wallace's water heater is 40 gallons and it's 3/4 full, calculate
    the total number of gallons of water they both have if Catherine's water heater is
    also full with water to 3/4 of its capacity.
Question 4.1: How much water is in Wallace's water heater?
New question 4.2: How much water do they have in total?
Is the new question useful? No. It is too hard to answer the new question based on the
    current information.

Question 5: [current question]
```

B MCTS Example

This section provides a detailed walkthrough of how RAP applies Monte Carlo Tree Search to solve a mathematical reasoning problem. We use a simple example from GSM8K to illustrate the key components and decision-making process of the algorithm.

B.1 Problem Statement

Consider the following problem from GSM8K:

> A robe takes 2 bolts of blue fiber and half that much white fiber. How many bolts in total does it take?

B.2 Search Process

The MCTS algorithm explores this problem through four key phases:

1. **Selection:** Starting from the root node, the algorithm selects child nodes using the UCT formula until reaching a leaf node. The selection balances between:
   - Exploiting known good paths (high Q-values)
   - Exploring uncertain paths (high uncertainty term)
2. **Expansion:** At the leaf node, RAP generates multiple candidate questions using the LLM, creating new nodes for exploration.
3. **Simulation:** From each new node, the algorithm simulates complete paths to terminal states, using simplified rewards for efficiency.
4. **Backpropagation:** Rewards from terminal states are propagated back up the tree, updating Q-values along the path.

B.3 Search Tree Visualization

Figure 2 illustrates the search tree developed during problem-solving. Note that for clarity, we show only the explored paths while omitting unexplored candidates. In practice, at each node, RAP generates multiple candidate questions (typically 4-5), but only the most promising ones according to the UCT formula are selected for further exploration. This means the actual search space is considerably larger than what is visualized here.

- **Node Structure:** Each node contains:
  - The question asked
  - The answer generated
  - Visit count
  - Fast reward (from lightweight evaluation)
  - Full reward (including state consistency)
- **Path Development:** The color coding indicates:

- Gray: Root node
- Blue: Intermediate reasoning steps
- Green: Solution nodes

B.4 Best Path Analysis

The final solution path demonstrates RAP's systematic problem decomposition approach. Each step builds upon previous knowledge while maintaining coherent reasoning:

```
BEST PATH:
==========================================================
Question: How much blue fiber does the robe need?, Reward: 0.81
Answer: 2 bolts of blue fiber. The answer is 2.
==========================================================
Question: How much white fiber does the robe need?, Reward: 0.72
Answer: 1/2 of that much white fiber. So the robe needs 2 * 1/2 = 1 bolt of white fiber. The answer is 1.
==========================================================
Question: How many bolts in total does it take?, Reward: 0.77
Answer: 2 + 1 = 3 bolts. The answer is 3.
==========================================================
Question: How many bolts of blue fiber does it take to make 4 robes?, Reward: 0.77
Answer: 2 + 2 + 2 + 2 = 8 bolts of blue fiber. The answer is 8.
==========================================================
Question: Now we can answer the question: A robe takes 2 bolts of blue fiber and
half that much white fiber.  How many bolts in total does it take?, Reward: 1.00
Answer: 2 + 1 = 3 bolts in total. The answer is 3.
==========================================================
```

C Experimental Setup

To evaluate the computational efficiency of RAP compared to CoT, we used Llama 3.2 3B on a subset of 1000 samples from GSM8K. Our implementation of RAP follows the hyperparameter configuration of the original paper [3], with minor adjustments to accommodate computational constraints. This setup allows us to maintain comparability while focusing on analyzing token efficiency rather than maximizing absolute performance. Our implementation of RAP used the following hyperparameters:

- Action generation limit: 4 candidates per node
- Confidence sampling: 5 samples per state evaluation
- Tree depth limit: 6 steps
- Maximum response length: 200 tokens
- Sampling temperature: 0.8

Most parameters were chosen to match the original paper [3] for comparability, including action generation limit, depth limit, maximum response length, and temperature. The confidence sampling parameter, which determines how many samples are used to evaluate state consistency, was reduced from the original paper's value of 8 to 5. This adjustment was made due to computational constraints, as each additional sample incurs significant token overhead. Our preliminary experiments showed minimal performance difference between using 5 versus 8 samples, making this a reasonable trade-off between computational efficiency and reliability.

D Statistical Analysis

To validate our findings statistically, we computed standard errors for all accuracy measurements using $SE = \sqrt{\frac{p(1-p)}{n}}$, where p is the measured accuracy and $n = 1000$ is our sample size. Figure 3 shows the complete performance comparison with error bars, indicating that RAP's performance advantages are statistically significant at iterations 1 and 7, while the apparent performance dip at iteration 5 (28.9% vs 30.1%) falls within our margin of error.

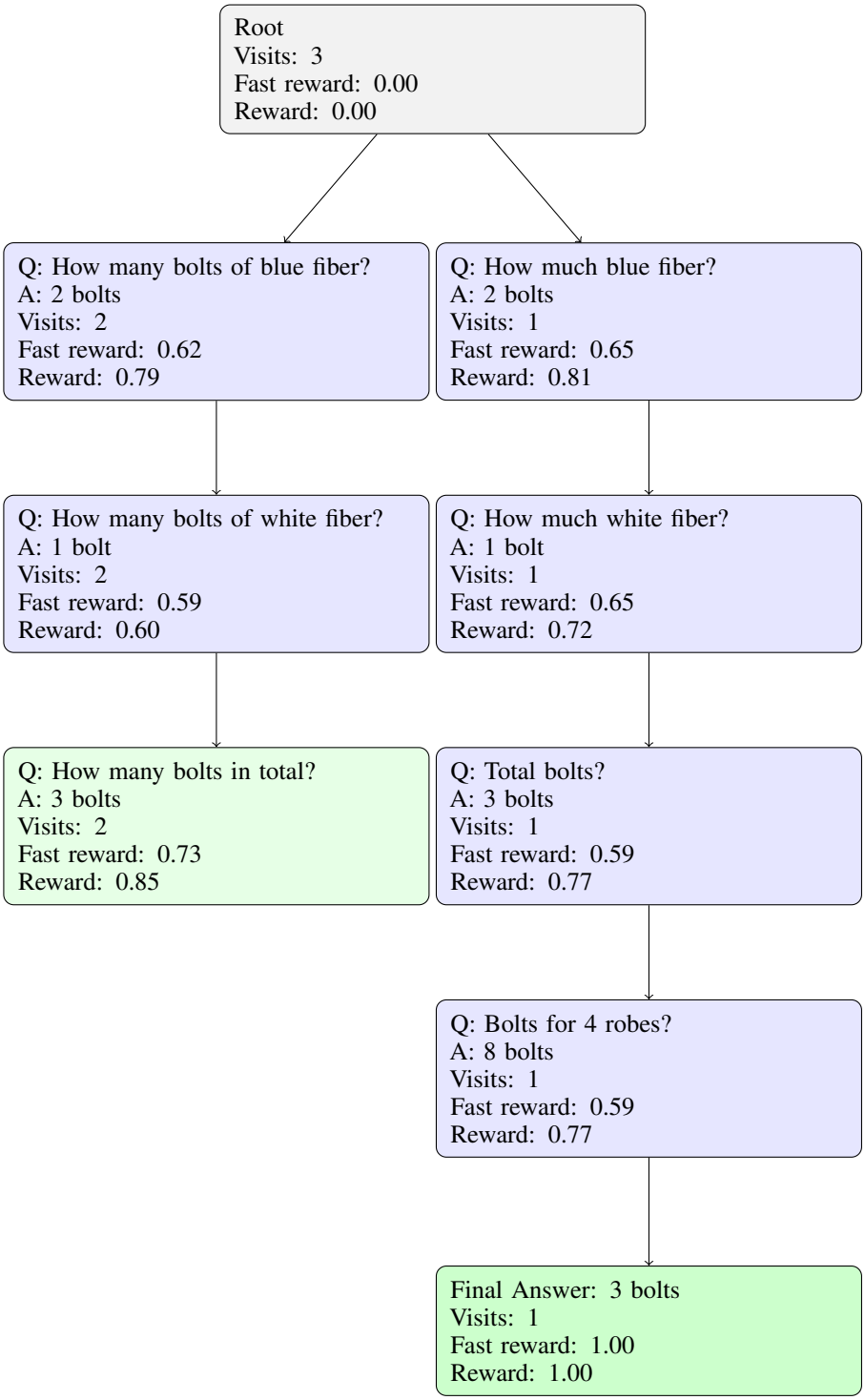


Figure 2: MCTS search tree showing the main paths explored. Node colors indicate progress (gray: root, blue: intermediate steps, green: solution nodes). Each node shows the question asked (Q), answer generated (A), number of visits, and both fast and final rewards.

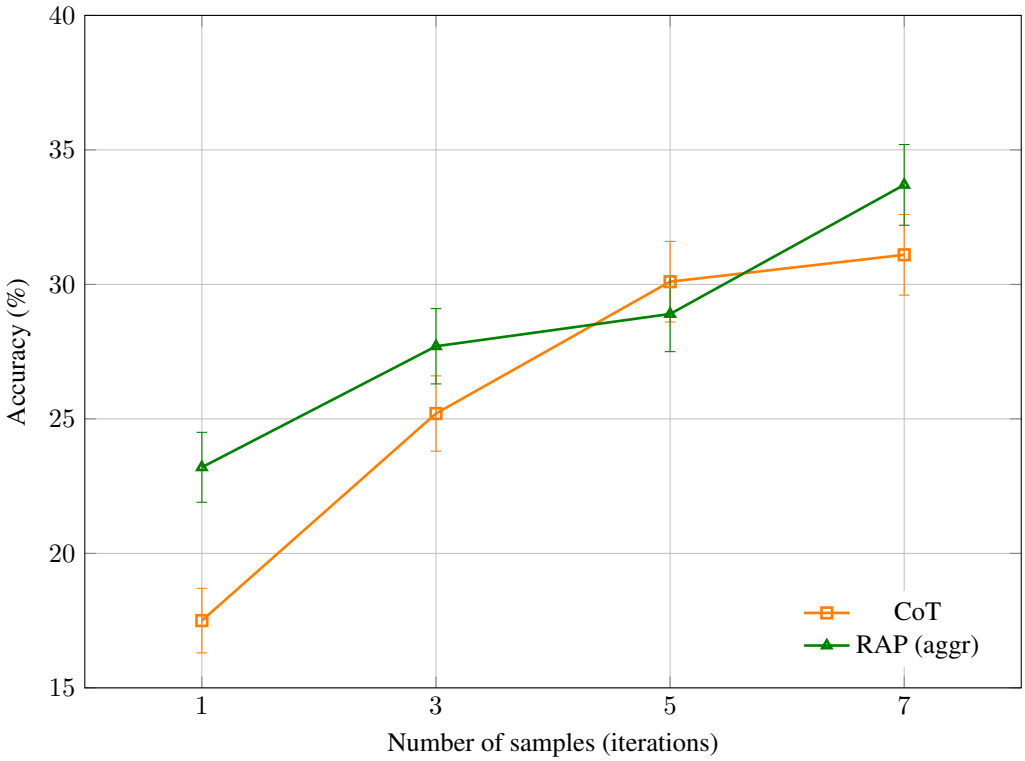


Figure 3: Extended analysis of RAP and CoT performance including standard errors ($n = 1000$ samples). Error bars indicate one standard error above and below the mean.

E Limitations and Future Work

Our evaluation focused solely on the GSM8K benchmark using Llama 3.2 3B, and our analysis doesn't account for potential optimizations that could reduce RAP's computational overhead. These limitations suggest promising directions for future work, particularly in exploring RAP's efficiency-performance tradeoffs across different tasks and model scales, and in developing techniques to reduce its token overhead while maintaining its ability to surpass traditional reasoning methods' performance ceilings.

Additional areas for future research include:

- Investigating RAP's performance on other reasoning tasks beyond mathematical word problems
- Exploring optimization techniques to reduce token overhead without sacrificing performance
- Analyzing the impact of model scale on RAP's efficiency-performance trade-offs