Efficient Reasoning Trajectory Traversal using a Modified Monte Carlo Tee Search

Andrew Zheng
University of Maryland, College Park
azheng15@umd.edu

Yubin Qin
University of Maryland, College Park
yubinq@umd.edu

Ishan Tamrakar
University of Maryland, College Park
itamraka@umd.edu

Abstract

Monte Carl Tree Search (MCTS) has shown promise as an exploratory mechanism for Large Language Model (LLM) and Vision Language Model (VLM) reasoning traces. However, the full MCTS algorithm is very expensive, requiring long and noisy full simulations to terminal states to produce a reward signal. As a result, using MCTS to explore more difficult and longer reasoning problems are simply intractible. In this paper, we propose a modified and budgeted Monte Carlo Tree Search (MCTS) module to guide the model response generation. More specifically, this framework provides a novel approach for self-improvement of models by generating reward traces without full MCTS simulations. We show that our methods increase efficiency tremendously without sacrificing accuracy. We provide code for our methodologies here.

1 Introduction

LLMs have displayed extraordinary capabilities in complex reasoning tasks. However, LLMs suffer from hallucinations, where models generate plausible but incorrect reasoning traces. These limitations have stalled the creation of superhuman intelligence, with efforts diverted to the understanding and mitigation of hallucinations in LLM tasks.

Previous works have found indicators of LLM hallucinations in reasoning tasks. Multiple sources have identified answer uniqueness as a reliable indicator of LLM response correctness [1] [2] [3]. Several frameworks have been proposed to utilize the answer uniqueness indicator. For example, Xue et. al. [4] considers the use of uncertainty quantification for VLM reasoning and benchmark construction. Zhang et. al [5] explores the use of token-level uncertainty for self-assessment and improvement. In addition, other prior works have asserted that models already contain a surplus of knowledge, and finetuning does not inject novelties [6] but rather makes certain phrases more likely. These results suggest that an efficient token-level search combined with uncertainty quantification is promising for future work. Search methods such as MCTS in combination with reinforcement learning have shown superhuman capabilities in chess and go [7]. Inspired by this success, recent works have focused on the application of these search methods to LLMs [8]. Most notably, AlphaLLM [9] proposes a model-driven self-improvement loop using MCTS search of best batch-of-token generations. Despite the powerful search capabilities of MCTS, the search process is incredibly inefficient, with each node generation iteration of the search requiring the generation of an entire response. This suggests the following: How can one design an efficient MCTS search capable of achieving comparable performance to previous MCTS-inspired search strategies?

Preprint. Under review.

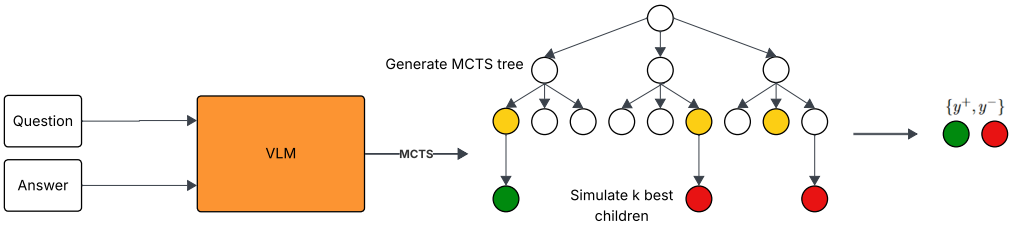

Figure 1: The self-improvement loop of our model for generating preference pairs for reinforcement learning. Given a question-image-answer pair, we use the VLM itself to generate reasoning traces using MCTS. Then, we take the most promising children, simulate to end, and generate preference pairs.

In this paper, we propose a modified but efficient MCTS-inspired search for model-guided reasoning. Our search module requires no full MCTS simulation rollouts for reward. Instead, we propose heuristics aligned with non-circular reasoning and conciseness. Similar to previous works, we search for the best response sentences at a time. To increase model efficiency and response conciseness, we implement a strict limit on MCTS tree size and we implement early stopping. We show a visualization of our MCTS-inspired search in figure 1. We present experimental results to show that our efficient MCTS-inspired search can achieve comparable performance to previous methods while being faster for reasoning questions.

2 Related Work

Much of the recent work on reasoning for VLMs have been on the curation of high-quality data for VLM training. ThinkLite-VL [10] presents a methodology for filtering out data points through MCTS, where data that is more challenging for VLMs were kept. More specifically, the more MCTS reasoning steps that need to be taken, the more challenging the data is. Chen et al. [11] focused on a data-curation pipeline that would improve the spatial reasoning capability of VLMs. This pipeline scrapes images from the internet and creates questions simple to humans which test the spacial reasoning of models. Liu et al. [12] curates data similarly to [10] through an MCTS process except the MCTS was used to generate reasoning traces instead of quantifying data difficulty. These reasoning traces were then fed into an actor-critic model, where a critic model was trained to provide good critics to an actor model to get the correct solution. Tian et. al [9] explores the use of MCTS for self-improvement of LLM responses. It is natural to consider a state-traversal algorithm where MCTS guides the search of the module. Kuznetsov [13] explores the use of MCTS to guide traversal in settings where the action space is Euclidean and continuous, whereas previous state of the art only used random noise to guide traversal. Shunyu Yao et. al [14] introduced the Tree of Thoughts framework which generalizes over Chain of Thought and allows a deliberate reasoning process. They do so by using a state evaluator heuristic to value which states to select, and primarily focus on problems where the size of the tree can be limited. Acuna et. all [8] uses an MCTS-inspired algorithm that injects subquestion-subanswer pairs into the model's output stream at test-time to generate better answers. In this framework, each action would be a subquestion that MCTS would traverse over to reach a terminal state. Many other benchmarks were created to test the visual learning capabilities of VLMs [15][16][17]. Although some works are similar, none of these works utilize modified and more efficient MCTS module to guide LLM responses.

3 Preliminaries

Consider a VLM model f_θ, and when given an input prompt $x := [x_1, x_2, ..., x_n]$, where each x_i is a token, generates a reasoning trace $y := [y_1, y_2, ..., y_m]$ containing answer $a := [a_1, ..., a_k]$. We denote the autoregressive, trainable policy $\pi_\theta(y \mid x) := \Pi_{i=1}^{L} p_\theta(y_i \mid x, y_{<i})$, where p_θ denotes a trainable probability distribution of tokens. A state s_t would represent the current incomplete response of a prompt, and the terminal state would be a full generated response of VLM f_θ. For our purpose, each action a_t would be the concatenation a new step of reasoning, either an equation or

sentence, to current response state s_t. The reward r_t would be the average reward of the simulations of a state s_t. We define the simulation R_t of reward function Our goal is as follows. We define the correctness function $C(x, y) \to \{0, 1\}$ by whether a terminal node, reached either by simulation or node generation, contains an answer a equivalent to ground truth answer. Given a VLM f_θ, find a response y such that $\pi_\theta(y \mid x) > 0$ and $C(x, y) = 1$. We discuss methodologies for searching for such a y in Section 4.

Monte Carlo Tree Search (MCTS) is a methodology for heuristic search that balances the exploration of unvisited states and the exploitation of visited promising path to find an optimal solution in complex search space without the needs to randomly explore all states. MCTS has been adapted for VLMs recently to treat reasoning as a sequential decision-making process. By structuring the reasoning steps as a search tree, MCTS allow the model to browse various logical path and evaluate the intermediate reasoning step before committing to the final answer. The standard MCTS algorithm mainly consist of four phases:

1. **Selection**
   The algorithm begins with the root node and traverse down to a leaf node. At each step, the tree use Upper Confidence Bound (UCB) to determine the next child node to visit, which mathematically balance the Exploitation and Exploration.
2. **Expansion**
   Once a leaf node is reached, if it is not a terminal state, the tree expands. For VLMs, the model will generate one or more new candidate reasoning steps extending from current contexts. These new steps are added as child nodes to the leaf.
3. **Simulation**
   Once a new leaf node is added, a rollout policy is used to estimate the potential value of reaching this state without performing exhaustive search. For VLMs, the model generate continuous reasoning steps until a stopping criterion is met, usually the end-of-sentence token.
4. **Backpropagation**
   Once the simulation reaches a terminal state, the reward is calculated based on whether or not the final answer is equivalent to the ground truth. This reward signal is propagated back up the tree, updating the value estimates (Q) and visit count (N) for every nodes on this path.

These four phases drive the search toward a desirable reasoning path. Crucially, the efficiency and success of this process highly depends on the Simulation and Backpropagation phases, where our Lite MCTS framework introduce its key modifications to enable efficient data collection.

4 Implementation

In this section, we introduce the entire modified MCTS module. First, we describe metrics for computing node similarity within our modified MCTS. This action similarity is integral for the reward function we utilize for shallow simulation components. We emphasize that our core contribution is the methodology for shallow rollouts. Previous results suggest that most models have the knowledge to solve most reasoning problems, and we show that a lighter MCTS can find correct reasoning traces.

4.1 Action Similarity

A crucial challenge in applying MCTS to generative models is the vastness of action space. Traditional use cases of MCTS explore problems with much smaller search spaces. To prevent the search from wasting resources on redundant or repetitive reasoning paths, we implement a node similarity metric to measure how similar two reasoning steps are. This enables our Lite MCTS framework to detect two important signals: Answer Convergence and Self-Looping Reasoning.

1. Answer Convergence: There are various ways to solve a problem, but there is only one correct answer. If distinct reasoning paths converge, the branches are more likely to be correct.

2. Self-Looping Reasoning: Repetition in reasoning is expensive and unnecessary. If a path contains similar reasoning steps, it might suffer from self-looping reasoning.

We utilize a Term Frequency-Inverse Document Frequency (TF-IDF) approach to quantify the sematic uniqueness of a generated reasoning step. For a given node n, let a_n be the action (reasoning step), and D is the collection of all generated action in current search tree. For each unique word w in a_n, we can calculate the Term Frequency (TF) and Inverse Document Frequency (IDF) as follows:

$$TF(w, a_n) = \mathbb{I}(w \in a)$$

$$IDF(w, D) = \log \frac{N}{\text{count}(w, D)}$$

where $\mathbb{I}$ is the indicator function, N is the number of nodes in the tree, and count(w, D) represent the number of nodes containing the word w. The TF-IDF vector can be obtained:

$$v = TF \times IDF$$

We measure the similarity between a new candidate action a_{new} and a set of existing nodes. To compute the consine similarity of these two TF-IDF vectorized representations v_{new} and v_{target}, the similarity score S is defined as follow:

$$S = \frac{v_{new} \cdot v_{target}}{||v_{new}|| \cdot ||v_{target}||}$$

We maintain a global counter that updates dynamically as the tree expands. Thus, we ensure the similarity metric evolves dynamically, penalizing self-looping reasoning and rewarding converging answers.

4.2 Reward Model

In our Lite MCTS framework, we prioritize breath over depth. Thus, we implement a piecewise heuristic reward function to effectively value each state. The reward is determined as follows:

1. **Terminal Correctness**: If the simulation reaches a terminal state, we verify the correctness of the generated answer with the ground truth.

$$R_{terminal} = \begin{cases} +1.0 & \text{generated answer is correct} \\ -1.0 & \text{generated answer is incorrect} \end{cases}$$

2. **Similarity**: If the state is not terminal, we utilize our similarity metric to access the structural quality of the reasoning path. We compare the new state with the existing search tree to detect two important signals:
   - **Penalty for self-looping reasoning**: If a new state is similar to a node within its own ancestor trace, it indicates self-looping reasoning. To discourage the model getting stuck in repetitive loop, we apply a penalty:

$$R_{loop} = -0.15$$

   - **Reward for converging answers**: If a new state is similar to a node in different branch, it indicates answer convergence. To encourage the model getting to this promosing reasoning, we apply a reward:

$$R_{converge} = +0.15$$

3. **Default Heuristic**: If the state is not terminal nor similar to others, we simply assign a neutral reward of 0. This encourage the tree search to continue exploring.

4.3 Shallow Simulations

The standard MCTS has a major flaw when applied to a language task. During the simulation phase, it is unclear how many rollout steps will be necessary to reach the terminal state. In other words, it has no upper bound. The simulation can be indefinite and results in huge computational bottleneck

during an iteration of MCTS. Furthermore, there is a need to simulate each node that we add to the tree till the terminal state in order to update the reward. The computational cost explodes when searching through a large space such as in the case of a reasoning task.
Shallow rollouts mean you stop at an earlier point before reaching the terminal state. Instead, you take only a few steps, estimate the value based on those few steps and return early with a reward. This reduces the computational cost dramatically. A simple way to enforce this would be to have a cap on how many steps to simulate. In Thinklite-VL, they use a fixed rollout depth of 10 for all problems with the possibility of early exit if terminal state is reached. This is still expensive as there is no guarantee that a terminal state is reached in 10 steps for hard problems. If a terminal state is not reached, it would not be possible to get a reward.
We propose an approach to get sound reward even with shallow rollouts. Using the same action similarity metric as described in 4.1, we are able to reduce the number of steps required during simulation to $k = 5$. The reward is updated based on the reward model described in 4.2.

Our simulation has three stopping conditions:

1. **Terminal state reached** (answer found with `\boxed{}` format):
   - Extract answer and check correctness against ground truth
   - Return $+1.0$ if correct, -1.0 if incorrect
   - Cache the rollout for later use in preference pair generation
2. **Similar state detected** (via TF-IDF cosine similarity > 0.9):
   - Check if similarity is within same trace or across different traces
   - **Circular reasoning** (same trace): Return -0.15 penalty for looping
   - **Convergent reasoning** (different traces): Return $+0.15$ reward for multiple paths reaching same conclusion
3. **Step limit reached** ($k = 5$ steps):
   - Return 0.0 neutral reward if no terminal or similar state found

4.4 Preference Pair Collection

After the construction of the MCTS-inspired search, we only collect preference-pairs of problems where a positive solution and a negative solution is reached. The existence of purely negative traces suggests that the problems would be too difficult for the model to solve, while only positive traces suggests the question was too easy. If even after reaching the node limit, we are not able to get terminal leaf nodes, we simulate the current response to finish to get the final answer from the model and get the reward based on whether an answer was found or not.

4.5 Model Choices

We make additional design choices that deviate from the standard MCTS design. While other works [9] suggest an adaptive branching methodology, we enforce a strict limit of $k = 5$ generations from each node for simplicity. We emphasize that this does not imply that each child has $k = 5$ children; each generation is checked for similarity with other nodes. In addition to a generation cap per child, we enforce a node cap to the entire tree. When this node cap is reached, we take the best $N = 5$ leaf nodes and simulate these nodes to terminal state. To prevent unnecessary searching, we implement early stopping when the model achieves the correct answer.

5 Experiments

5.1 Experimental Setup

For experiments of the efficient MCTS-inspired search we propose, we choose to evaluate on a random 100-sample subset of MathVista [18]. We provide the data sample ids used in the Appendix A.2. We compare the performance of our methodology to the standard MCTS, with a couple modifications. Similar to the MCTS-inspired search, we implement the standard MCTS to produce a fixed k children nodes. We also implement early stopping. As a consequence, the only difference between the two

methods are the similarity node matching modules. To guide MCTS generation, we use Qwen2-VL-7B for both iterations of MCTS. We set $c = 0.5$ for the expansion constant of UCB. For each node in the tree, we generate 7 children, and we fix simulation rollouts to be of max depth 5. We compare the two methods in terms of efficiency and accuracy.

5.2 Results

Here, we provide comparisons between the standard MCTS and the efficient MCTS. The accuracy values and the time taken are **average** values.

Table 1: Comparison between standard MCTS and efficient MCTS

Model	Accuracy	Time Taken (sec)
Standard	55%	344.01
Our Method	**57%**	**315.56**

We emphasize that the MathVista questions require few reasoning steps, so for more complex questions, we expect greater speedup. These results show our additions of similar response filtering for shallow rollouts increase efficiency while maintaining accuracy.

6 Conclusion

In this paper, we have described a more efficient modification of MCTS to perform state search over model generation possibilities for self-improvement. In addition, we showcase a small dataset generated with Qwen2-VL-7B using our light MCTS methodology. Experimentally, we show that our methodology achieves comparable performance while being more efficient compared to its traditional counterparts. A natural extension of our work would be to consider the results of our efficient MCTS-inspired search on more datasets. For our next iteration of this work, we target GSM8k [19], ImageVQA [20], MATH [21], and AIME datasets. Then, we seek to generate more data to finetuning Qwen2-VL-7B. A further extension of this work would be to consider the use of this framework to compute sequential finetuning. We hope future works address the following: 1) Can a model be self-improved in a way that can allow it to solve more complicated questions that require deeper reasoning traces? 2) How much improvement does each iteration of our methodology improve the model? What sequence of curated data is optimal to drive this self-improvement look? We plan to visit these questions after the submission of our course project.

Our methodology produces one positive and many negative pairs for each question. We hope that future works can consider the correction of a negative trace given the correct answer and a positive trace. We hypothesize that a distribution of positive traces and negative traces would help the models learn reasoning more robustly and less noisy. The benefit of such methodology is unknown, and we leave these theories to future works.

References

[1] M. Xiong, A. Santilli, M. Kirchhof, A. Golinski, and S. Williamson, "Efficient and effective uncertainty quantification for llms," in *Proceedings of the NeurIPS Safe Generative AI Workshop 2024*, 2024. [Online]. Available: https://openreview.net/pdf?id=QKRLH57ATT

[2] X. Wang, J. Wei, D. Schuurmans, Q. V. Le, E. H. Chi, S. Narang, A. Chowdhery, and D. Zhou, "Self-consistency improves chain of thought reasoning in language models," *arXiv preprint arXiv:2203.11171*, 2022. [Online]. Available: https://arxiv.org/pdf/2203.11171

[3] L. Liu, Y. Pan, X. Li, and G. Chen, "Uncertainty estimation and quantification for llms: A simple supervised approach," *arXiv preprint arXiv:2404.15993*, 2024. [Online]. Available: https://arxiv.org/pdf/2404.15993

[4] Q. Xue, W. Liu, S. Wang, H. Wang, Y. Wu, and W. Gao, "Reasoning path and latent state analysis for multi-view visual spatial reasoning: A cognitive science perspective," *arXiv preprint arXiv:2512.02340*, 2025. [Online]. Available: https://arxiv.org/pdf/2512.02340

[5] T. Zhang, H. Shi, Y. Wang, H. Wang, X. He, Z. Li, H. Chen, L. Han, K. Xu, H. Zhang, D. Metaxas, and H. Wang, "Token-level uncertainty estimation for large language model reasoning," *arXiv preprint arXiv:2505.11737*, 2025. [Online]. Available: https://arxiv.org/pdf/2505.11737

[6] J. Merullo, N. A. Smith, S. Wiegreffe, and Y. Elazar, "On linear representations and pretraining data frequency in language models," in *Proceedings of the International Conference on Learning Representations (ICLR) 2025*, 2025. [Online]. Available: https://openreview.net/pdf?id=EDoD3DgivF

[7] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017. [Online]. Available: https://arxiv.org/abs/1712.01815

[8] D. A. et. al, "Socratic-mcts: Test-time visual reasoning by asking the right questions," *arXiv preprint arXiv:2506.08927*, 2025.

[9] Y. T. et. al, "Toward self-improvement of llms via imagination, searching, and criticizing," in *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*, 2024. [Online]. Available: https://arxiv.org/abs/2404.12253

[10] X. W. et. al, "Sota with less: Mcts-guided sample selection for data-efficient visual reasoning self-improvement," *arXiv preprint arXiv:2504.07934*, 2025.

[11] e. a. Boyuan Chen, "Bspatialvlm: Endowing vision-language models with spatial reasoning capabilities," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[12] S. L. et. al, "Mmc: Iterative refinement of vlm reasoning via mcts-based multimodal critique," *arXiv preprint arXiv:2504.11009*, 2025.

[13] I. Kuznetsov, "Guided exploration in reinforcement learning via monte carlo critic optimization," *arXiv preprint arXiv:2206.12674*, 2024.

[14] S. Y. et. al, "Tree of thoughts: Deliberate problem solving with large language models," *NeurIPS 2023*, 2023.

[15] I. H. et. al, "Conme: Rethinking evaluation of compositional reasoning for modern vlms," *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*, 2024.

[16] A. N. et. al, "Zero-shot visual reasoning by vision-language models: Benchmarking and analysis," *IEEE*, 2024.

[17] Y. Z. et. al, "Spinbench: Perspective and rotation as a lens on spatial reasoning in vlms," *arXiv preprint arXiv:/2509.25390*, 2025.

[18] P. Lu, H. Bansal, T. Xia, J. Liu, C. Li, H. Hajishirzi, H. Cheng, K.-W. Chang, M. Galley, and J. Gao, "Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts," in *International Conference on Learning Representations (ICLR)*, 2024.

[19] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano *et al.*, "Training verifiers to solve math word problems," *arXiv preprint arXiv:2110.14168*, 2021. [Online]. Available: https://arxiv.org/abs/2110.14168

[20] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. Lawrence Zitnick, and D. Parikh, "Vqa: Visual question answering," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 2425–2433.

[21] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt, "Measuring mathematical problem solving with the math dataset," *arXiv preprint arXiv:2103.03874*, 2021. [Online]. Available: https://arxiv.org/abs/2103.03874

A Supplementary Material

We now describe additional material to support claims we make in this paper.

A.1 Examples of MCTS Traces

We present an explicit example of our work.

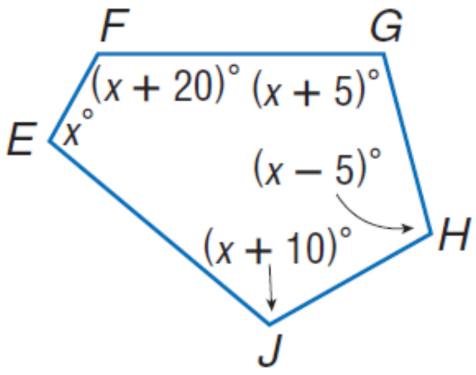

Figure 2: Find Angle H

In the following generation, the model achieves the wrong answer:

Given: The sum of the interior angles of a polygon.

In a pentagon, the sum of the interior angles is \(540^\circ\).

The interior angles are given as \(x^\circ, (x+20)^\circ, (x+5)^\circ, (x-5)^\circ\), and \((x+10)^\circ\).

Set up the equation:
\[x + (x+20) + (x+5) + (x-5) + (x+10) = 540\]

Combine like terms:
\[5x + 40 = 540\]

Hallucinates!

Subtract 40 from both sides:
\[5x = 500\]

Divide both sides by 5:
\[x = 100\]

So, the angle \(x\) is \(100^\circ\).

Plug \(x = 100^\circ\) into each angle to find the angle at \(H\):
\[x = 100^\circ\]
\[(x+20)^\circ = 100 + 20 = 120^\circ\]
\[(x+5)^\circ = 100 + 5 = 105^\circ\]
\[(x-5)^\circ = 100 - 5 = 95^\circ\]
\[(x+10)^\circ = 100 + 10 = 110^\circ\]

Therefore, the angle at \(H\) is \(\boxed{95^\circ}\).

In the following trace, the model achieves the correct answer concisely:

To find the measure of angle H in the given pentagon, we can use the fact that the sum of the interior angles of a pentagon is 540°.\n\nLet's denote the unknown angles in the pentagon as follows:\n- \\(F \\) = \\[(x + 20)^\\circ\\]\n- \\(G \\) = \\[(x + 5)^\\circ\\]\n- \\(E \\) = \\[x^\\circ\\]\n- \\(H \\) = \\[(x - 5)^\\circ\\]\n- \\(J \\) = \\[(x + both sides:\n\\[510° = 5x \\]\n\nDivide by 5:\n\\[x = 102 \\]\n\nNow that we have \\[x \\], we can find the measure of angle H:
\[H = (x - 5)° \]
\[H = (102 - 5)° \]
\[H = 97° \]

A.2 Trace ID samples from MathVista

We use the following sampled trace ids from MathVista for comparison:

traceids = [7, 26, 28, 31, 33, 45, 47, 72, 81, 82, 90, 95, 96, 100, 104, 105, 115, 128, 143, 160, 164, 167, 197, 204, 215, 221, 224, 226, 229, 234, 239, 251, 271, 282, 285, 297, 301, 345, 349, 353, 364, 368, 371, 374, 380, 388, 390, 430, 433, 460, 465, 471, 518, 550, 559, 566, 575, 592, 604, 605, 617, 619, 634, 644, 651, 655, 666, 678, 687, 693, 715, 719, 722, 734, 748, 755, 759, 760, 778, 782, 792, 826, 827, 829, 850, 855, 868, 876, 883, 888, 891, 907, 914, 948, 955, 965, 979, 984, 987, 991].