MDToC: Metacognitive Dynamic Tree of Concepts for boosting mathematical problem-solving of large language models

Tung Duong Ta[1]*, Tim Oates[1], Thien Van Luong[2], Huan Vu[2], Tien Cuong Nguyen[3]

[1] *Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County, 1000 Hilltop Circle, Baltimore, Maryland 21250, USA*

[2] *National Economics University, 207 Giai Phong Road, Bach Mai Ward, Hanoi, Vietnam*

[3] *VNPT AI, Lane 55 Huynh Thuc Khang Street, Yen Hoa Ward, Hanoi, Vietnam*

Received 2 July 2026; revised 31 August 2026; accepted 4 September 2026

Abstract:

Despite advances in mathematical reasoning capabilities, Large Language Models (LLMs) still struggle with calculation verification when using established prompting techniques. Chain-of-Thought prompting and its self-consistency variant lack evaluation of intermediate reasoning steps. Hierarchical techniques such as Tree-of-Thoughts (ToT) and Graph-of-Thoughts (GoT) evaluate abstract thoughts, which are difficult to score reliably. We present MDToC (Metacognitive Dynamic Tree of Concepts), a three-phase prompting technique built on the metacognitive stages of planning, monitoring, and reviewing. MDToC constructs a concept tree, develops accuracy-verified calculations for each concept, and employs majority voting to evaluate competing solutions. Evaluations across CHAMP, MATH, and Game-of-24 benchmarks demonstrate our MDToC's effectiveness, with GPT-4-Turbo achieving 58.1% on CHAMP, 86.6% on MATH, and 85% on Game-of-24, outperforming GoT by 5%, 5.4%, and 4% on all these tasks, respectively, without hand-engineered hints. MDToC consistently matches or surpasses existing prompting methods across all backbone models, yielding improvements of up to 19% over ToT and 13% over GoT. In addition, delegating auxiliary components to lighter models reduces cost while maintaining accuracy. These results establish metacognitive calculation verification as a promising direction for enhanced mathematical reasoning.

Keywords: **large language models, metacognition, mathematical reasoning, problem-solving, tree of concepts.**

1. Introduction

Large Language Models (LLMs) like GPT-4 [1] and Claude [2] demonstrate proficiency in various mathematical problems, excelling in easy to medium-difficulty tasks as evidenced by their performance on benchmarks such as GSM8k [3] and SVAMP [4]. However, their efficacy diminishes when faced with complex challenges presented in datasets such as MATH [5] and CHAMP [6]. In these demanding scenarios, LLMs often struggle with accurate multi-step reasoning and solution derivation. A key factor in this performance degradation is the models' propensity for errors in intermediate calculations and logical deductions [7, 8]. These compounding inaccuracies result in poor performance on datasets featuring hard mathematical problems, opening a critical area for improvement in the multi-step reasoning capabilities of LLMs. Recent surveys of mathematical reasoning with LLMs likewise identify the verification of intermediate calculations as a central open challenge [9].

Researchers have widely adopted prompting techniques, particularly Chain-of-Thought (CoT) [10] and self-consistency CoT (SC-CoT) [11], to enhance LLMs' multi-step reasoning capabilities without additional training. These methods enable models to decompose complex reasoning processes into smaller steps, improving overall accuracy. In particular, CoT encourages articulation of thought processes, while SC-CoT generates multiple demonstrations with majority voting. However, these approaches have limitations: CoT may constrain diverse

* *Corresponding author: Email: dta1@umbc.edu*

CC BY NC ND

Crossref Similarity Check Powered by iThenticate

problem-solving pathways, while SC-CoT lacks crucial evaluation of intermediate reasoning steps. This can lead to erroneous samples and inaccurate voting outcomes. As a result, using these prompting techniques, advanced LLMs such as GPT-4 suffer from poor performance. For example, GPT-4 with SC-CoT achieves only 9% accuracy on the Game-of-24 task [12]. Therefore, it is essential to develop more robust reasoning methodologies.

Recent hierarchical prompting techniques like Tree-of-Thoughts (ToT) [12, 13] and Graph-of-Thoughts (GoT) [14, 15] have advanced reasoning capabilities through structured thought representation and intermediate evaluation, achieving impressive results on complex tasks (74% accuracy on Game-of-24 [12] and 89% on Sequence-Sorting-64-elements [14], respectively). However, as shown in Fig. 1, these approaches suffer from ill-defined evaluation criteria for diverse thought forms (mathematical analysis, concepts, calculations), leading to heavy reliance on powerful LLMs like GPT-4 that produce approximated and unreliable evaluation scores. This standardization challenge creates vulnerabilities in the critical processes of thought selection and connection pruning, while the need for domain-specific customization limits generalizability across different problem types.

Amidst the numerous successes of the CoT, ToT, and GoT prompting techniques, there have been several explorations of cognitive prompting methods for mathematical problem-solving [16]. In the field of psychology, metacognition enables individuals to reflect on and critically analyze their thought processes [17]. Recent research has enhanced model capabilities with metacognitive processes for natural language understanding tasks. For example, Y. Wang, et al. (2023) [18] demonstrated that LLMs prompted with metacognitive thinking outperformed previous techniques such as zero-shot [19, 20] or CoT prompting [10] across various NLP tasks. Y. Zhou, et al. (2024) [21] highlighted the effectiveness of the metacognitive approach in improving the retrieval-augmented generation process for LLMs. However, the application of metacognition to mathematical problem-solving remains relatively unexplored, with notable exceptions such as A. Didolkar, et al. (2024) [22], who showed that metacognitive approaches enhance mathematical reasoning in LLMs by reflecting on clustered math skills and thereby providing relevant in-context examples. Our work extends this research direction by applying metacognition to LLMs for solving mathematical problems.

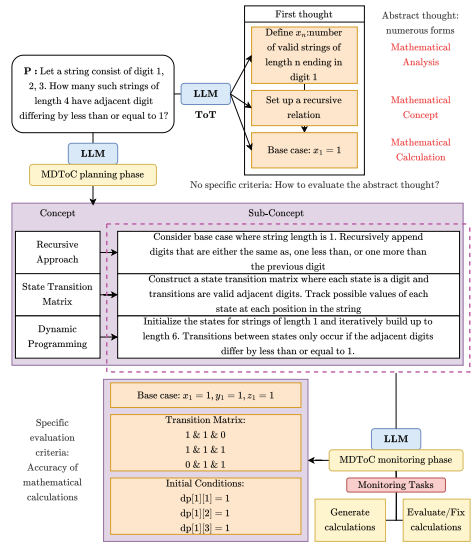


Fig. 1. ToT prompting yields initial abstract thoughts (e.g., analyses, concepts, calculations; in red), which are challenging to evaluate due to the intangible nature of conceptual reasoning and the lack of specific criteria to measure their correctness or completeness. Our MDToC addresses these abstract thoughts by first generating concrete concepts and then producing relevant calculations for those concepts. We only evaluate the preciseness of the calculations through mathematical accuracy checks, enabling precise evaluation and thus improving problem-solving reliability.

Motivated by the aforementioned background, we propose MDToC (Metacognitive Dynamic Tree of Concepts), a novel three-phase prompting technique that transforms abstract thoughts into concepts and evaluable calculations. MDToC employs a depth-two concept tree in the planning phase to explore diverse mathematical concepts while constraining the solution space, followed by a monitoring phase that expands sub-concepts with calculation steps using four specialized LLMs, and concludes with a reviewing phase utilizing majority voting following the self-consistency voting mechanism [23]. This comprehensive framework has demonstrated significant effectiveness, with GPT-3.5+MDToC achieving 39.5% accuracy on CHAMP (outperforming GPT-3.5 with annotated concepts by 5.1%) and GPT-4o-mini+MDToC attaining 75% accuracy on Game-of-24 (surpassing GPT-4o-mini with ToT by 19%).

2. Materials and methods

This section first reviews common prompting techniques and related work on self-feedback and training-based reasoning. It then describes the metacognitive foundation and the proposed MDToC.

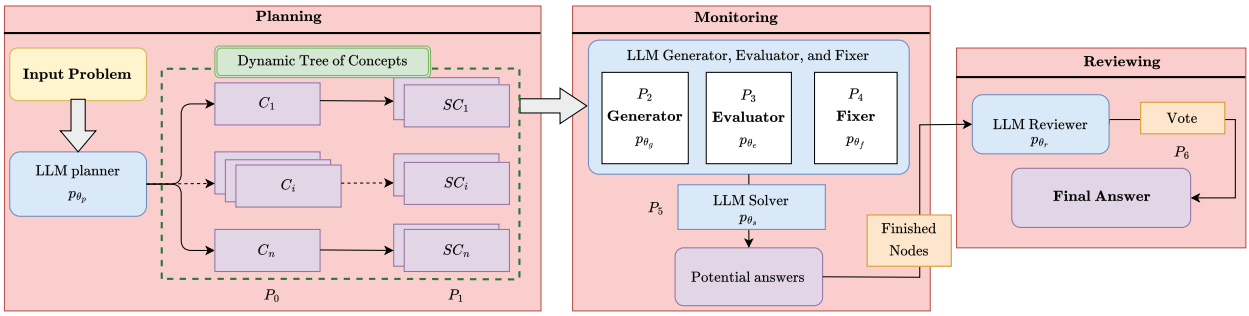


Fig. 2. Proposed MDToC prompting structure. C represents the first-depth concept, while SC represents the second-depth sub-concept. P_0 and P_1 are prompts used in the planning phase shown in Fig. 3, while P_2, P_3, P_4, and P_5 are prompts used in the monitoring phase given in Fig. 4. Prompt P_6 is used in the reviewing phase, as shown in Fig. 6.

2.1. Prompting techniques

Let p_θ be an LLM parameterized by θ. Three of the most common prompting techniques for mathematical reasoning with p_θ are described below.

1. **Chain-of-Thought**. Given an input problem x, the LLM is guided through a sequence c of reasoning steps to produce an answer y [10]. Specifically, the sequence c of reasoning steps is generated by the LLM based on the input problem, expressed as $c \sim p_\theta(c|x)$. The final answer y is then generated by the LLM based on both the input problem and the sequence of reasoning steps expressed as $y \sim p_\theta(y|x, c)$.
2. **Tree-of-Thought**. Given an input problem x, the LLM navigates a tree structure where each node i represents a state $s = [x, z_{1...i}]$, with $z_{1...i}$ denoting a sequence of thoughts along the current path [12]. The LLM generates a new thought z_{i+1} based on the current state s, expressed as $z_{i+1} \sim p_\theta(z_{i+1}|x, z_{1...i})$. A new node with the state $s' = [x, z_{1...i+1}]$ is then appended to the current node i in the tree.
   Each state s in a set of states S undergoes evaluation through either numerical values or voting to determine the viability of further path exploration. The numerical evaluation $V(p_\theta, s)$ is expressed as $V(p_\theta, s) \sim p_\theta(v|s)\, \forall s \in S$, where v is the numerical value. The voting evaluation $V(p_\theta, s)$ is expressed as $V(p_\theta, s) = 1[s = s^*]$ where $s^* \sim p_\theta^{vote}(s^*|S)$. In this context, the LLM votes for state s^* given the set of states S, employing the indicator function $1[s = s^*]$ to determine whether a state s corresponds to the voted state s^*.
3. **Graph-of-Thought** Given an input problem x, the LLM navigates a directed graph structure $G = (V, E)$, where V is the set of vertices representing thoughts, and $E \subseteq V \times V$ is the set of edges representing dependencies among thoughts [14]. Denote that V^+ and E^+ represent newly added vertices and edges, while V^- and E^- denote removed vertices and edges, respectively. Unless stated otherwise, $V^- = E^- = \emptyset$. The graph of thoughts is manipulated through three primary operations: aggregation, refinement, and generation.
   Following these operations, the graph is updated as $G' = T(G, p_\theta) = (V', E')$, where $V' = (V \cup V^+) \backslash V^-$, $E' = (E \cup E^+) \backslash E^-$. Each node v in graph G is subsequently evaluated by the LLM using either a scoring or ranking method. The scoring function is expressed as $s = E(p_\theta, v, G) \sim p_\theta(s|v, G)$, where s denotes the score value of node v. Conversely, the ranking function is expressed as $\{v_1, v_2, ..., v_h\} = R(p_\theta, h, G) \sim p_\theta(\{v_1, v_2, ..., v_h\}|G, h)$, where h represents the number of top-ranking thoughts to be returned.

2.2. Iterative self-feedback and training-based Reasoning

Beyond hierarchical prompting techniques, a related line of work improves reasoning through iterative self-feedback. Self-Refine [24] prompts a single LLM to evaluate and refine its own response repeatedly, but the evaluation is applied to the whole response rather than to intermediate calculations. In contrast, the monitoring phase of our MDToC uses LLMs to assess and correct only the intermediate calculations. Furthermore, it limits repeated evaluation and regeneration to two attempts per calculation. Recent findings support this design. Self-correction by LLMs without external feedback is unreliable [25]. Moreover, LLMs frequently re-evaluate calculations that are already correct. This increases token consumption without detecting new errors [26].

Whereas these methods, like MDToC, operate at inference time, another line of work improves

mathematical reasoning through training. For example, DeepSeekMath [27] continues pretraining open-source LLMs on mathematical data and applies policy optimization. More recently, a verifier has been trained to produce feedback for each reasoning step [28]. Surveys document rapid progress in fine-tuning and reinforcement learning for mathematical reasoning [9]. Our MDToC requires no additional training and can be applied to any backbone model, including such trained LLMs.

2.3. Metacognition

Metacognition, the ability to reflect on and regulate one's thought processes, plays a crucial role in advanced problem-solving and decision-making. It serves as an overarching framework guiding the effective application of cognitive strategies. This study aims to endow language models with a simulated metacognitive process, mimicking the human capacity for "thinking about thinking". Our MDToC approach employs a hierarchical prompting structure incorporating three foundational stages of metacognition: planning, monitoring, and reviewing [29], specifically designed for mathematical problem-solving.

The planning stage creates a conceptual roadmap by establishing strategies and approaches. During monitoring, we implement a metacognitive mechanism that enables self-evaluation and correction of calculations in progress. The final reviewing stage examines solutions, filtering out empty results and identifying the most frequently occurring valid answer.

2.4. The proposed MDToC

This research introduces a novel prompting approach, called MDToC, utilizing a dynamic tree of concepts within a tripartite metacognitive framework of planning, monitoring, and reviewing. This method addresses limitations in existing hierarchical prompting techniques for LLMs, such as unreliable evaluations of abstract thoughts and lack of generalizability. Our approach enables the exploration of diverse reasoning paths and selects the final solution through majority voting. Fig. 2 shows a visual representation of our methodology.

2.4.1. Planning

During the initial planning phase, we construct a concept tree $T = (V, E)$ with a depth of two where V is a set of nodes and E is a set of edges. We begin by instructing our LLM planner p_{θ_p} with prompt P_0 to extract the objective of the question, called q, and generate n distinct concepts $\{c_1^{d=1}, c_2^{d=1}, ..., c_n^{d=1}\} \sim p_{\theta_p}(q)$, where $d = 1$ represents the first depth of the tree. Each i-th concept is articulated as a detailed sentence, providing a mathematical or programmatic response to q. We then incorporate n concepts as nodes within the tree structure, where $V = \{c_1^{d=1}, c_2^{d=1}, ..., c_n^{d=1}\}$.

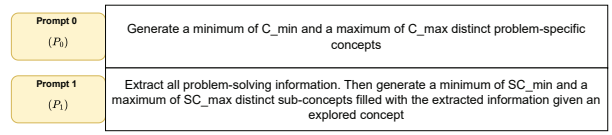


Fig. 3. Prompts for the planning phase.

For each i-th concept at depth $d = 1$, we further prompt our LLM planner p_{θ_p} with prompt P_1 to produce m distinct sub-concepts $\{c_{i1}^{d=2}, c_{i2}^{d=2}, ..., c_{im}^{d=2}\} \sim p_{\theta_p}(q, c_i^{d=1})$. These sub-concepts, each expressed in two detailed sentences, serve to elucidate and expand upon the i-th concept with question-solving information. Within our tree structure, we position each j-th sub-concept as a child node to its corresponding i-th concept. Fig. 3 demonstrates our prompts for P_0 and P_1. Our tree T is then expressed as:

$$V = \{c_1^{d=1}, ..., c_n^{d=1}, c_{11}^{d=2}, ..., c_{1m}^{d=2}, ..., c_{n1}^{d=2}, ..., c_{nm}^{d=2}\}$$
$$E = \{(c_1^{d=1}, c_{11}^{d=2}), ..., (c_n^{d=1}, c_{nm}^{d=2})\}$$

The construction of our dynamic tree T incorporates four key parameters: C_{min}, C_{max}, SC_{min}, and SC_{max}. These parameters define the concept range (C_{min} and C_{max}) and sub-concept range (SC_{min} and SC_{max}), as illustrated in prompts P_0 and P_1 of Fig. 3, respectively. The magnitude of these parameters directly correlates with the breadth of concept exploration, where larger values facilitate a more extensive conceptual landscape. Upon examining the annotated concepts in CHAMP [6] where each problem receives 3 hints, we decided to opt for a two-depth dynamic tree structure. We aim to prevent LLMs from excessively relying on generated concepts while fostering a more flexible problem-solving process that can adapt to various problem types and complexities.

2.4.2. Monitoring

In the monitoring phase, we employ a ToT-based structure [12] to solve a concept tree in t iterations. We denote a mathematical calculation as χ. Unlike the traditional ToT approach which samples k thoughts and selects a subset of them, our MDToC samples and evaluates all k mathematical calculations with two LLM components: an LLM evaluator p_{θ_e} and

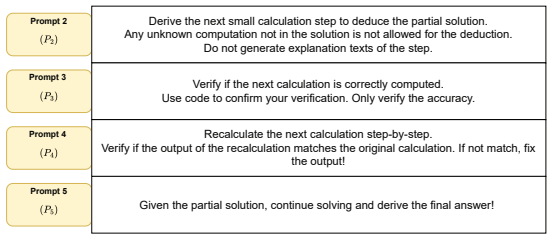


Fig. 4. Prompts for the monitoring phase.

an LLM generator p_{θ_g}. The generator p_{θ_g} prompted with P_2 samples k calculations, while the evaluator p_{θ_e} prompted with P_3 assesses the accuracy of each calculation. Specifically, when the evaluator p_{θ_e} identifies an error in a calculation, it returns a negative response ("No") accompanied by a detailed explanation of the error.

The k-th calculation $\chi_{ijk}^{d\geq 3}$ is sampled from the generator as $\chi_{ijk}^{d\geq 3} \sim p_{\theta_g}(\chi_{ijk}^{d\geq 3}|c_i^{d=1}, c_{ij}^{d=2}, \chi_{ijk}^{d-1\geq 3})$ where $\chi_{ijk}^{d-1\geq 3}$ represents previous calculations. The evaluation result $V(p_{\theta_e}, \chi_{ijk}^{d\geq 3})$ for the k-th calculation is expressed as $V(p_{\theta_e}, \chi_{ijk}^{d\geq 3}) \sim p_{\theta_e}(v_e, r_e|(c_i^{d=1}, c_{ij}^{d=2}, \chi_{ijk}^{d-1\geq 3}, \chi_{ijk}^{d\geq 3})$, where v_e is the binary result of 0 or 1 and r_e is the evaluation reason when $v_e = 0$. If $v_e = 0$, the generator regenerates the calculation $\chi_{ijk}^{d\geq 3} \sim p_{\theta_g}(\chi_{ijk}^{d\geq 3}|c_i^{d=1}, c_{ij}^{d=2}, \chi_{ijk}^{d-1\geq 3}, \chi_{ijk}^{d\geq 3}, r_e)$.

We further introduce an LLM fixer p_{θ_f} prompted with P_5 to fix the errors in the k-th calculation as $\chi_{ijk}^{d\geq 3} \sim p_{\theta_f}(\chi_{ijk}^{d\geq 3}|\chi_{ijk}^{d\geq 3})$. Specifically, after the complete cycle of generation and evaluation of calculations, p_{θ_f} addresses and corrects any remaining calculation errors. Subsequently, these k calculations are appended as nodes to the tree. After some iterations, we treat the current series of calculations as a partial solution and employ an LLM solver p_{θ_s} prompted with P_4 to resolve the partial solution. The solver's response is subsequently appended to the tree and marked as a "Finished Node", thereby terminating the exploratory process. Fig. 4 illustrates our prompts P_2, P_3, P_4, and P_5.

To monitor our concept tree with calculations, we introduce two additional parameters: c_s and t. c_s specifies the number of intermediate calculations generated at each iteration, while t represents the number of iterations. These parameters allow us to control the exploration behavior, balancing between breadth and depth.

Example analysis. Fig. 5 compares intermediate reasoning steps between GPT-3.5-Turbo with CoT + CH and GPT-3.5-Turbo with MDToC. It highlights challenges in recursive calculations requiring precise intermediate computations. Fig. 5A shows GPT-3.5-Turbo's performance is limited to accurate calculation of the base case only, with errors emerging in y_2 and cascading through subsequent steps. This leads to an incorrect final sum of 34.

Conversely, Fig. 5B illustrates the efficacy of our MDToC approach in mitigating computational errors. When applying MDToC, the likelihood of erroneous intermediate calculations is significantly reduced with 2 evaluation and regeneration attempts. This is evidenced by the correct computation of y_2, accurately determined as the sum of $x_1 = 1$, $y_1 = 1$, and $z_1 = 1$, yielding the correct result of 3. This precise evaluation of x_2, y_2, and z_2 serves as a crucial foundation, enabling accurate subsequent calculations for strings of lengths of 3 and 4, ultimately leading to the correct final answer of 41.

2.4.3. Reviewing

In the last phase, we obtain the results from the monitored tree and select the top-voted answers. Given a list of solutions marked by "Finished Node" as A, we use an LLM reviewer p_{θ_r} prompted with P_6 shown in Fig. 6 to conduct a majority voting of A. This stage involves finding the most common answer from A, returned as $\tilde{a} \sim p_{\theta_r}(A)$, where $\tilde{a}$ is the final solution to the objective q.

3. Results and discussion

3.1. Dataset

Our MDToC approach is evaluated using three datasets: CHAMP [6] (270 high school-level competition math problems across five categories, providing insight into concept tree-driven reasoning), MATH [5] (12,500 competition-level problems from sources like AIME [30] and AMC 10/12 [31] covering various advanced topics including algebra, geometry, and number theory), and a subset of 100 challenging Game-of-24 puzzles (indexed 901-1000, selected for direct comparison with ToT approaches) [12].

3.2. Parameters

Planning phase For the CHAMP and MATH datasets, we set $C_{min} = 2$, $C_{max} = 4$, $SC_{min} = 1$, and $SC_{max} = 2$ (see Section 2.4.1). In contrast, all parameters were set to 1 for Game-of-24 experiments, as these problems require diverse calculation combinations rather than varied mathematical concepts.

Monitoring phase For the CHAMP and MATH datasets, we use $c_s = 2$ and $t = 10$ (see Section 2.4.2), emphasizing in-depth concept decomposition

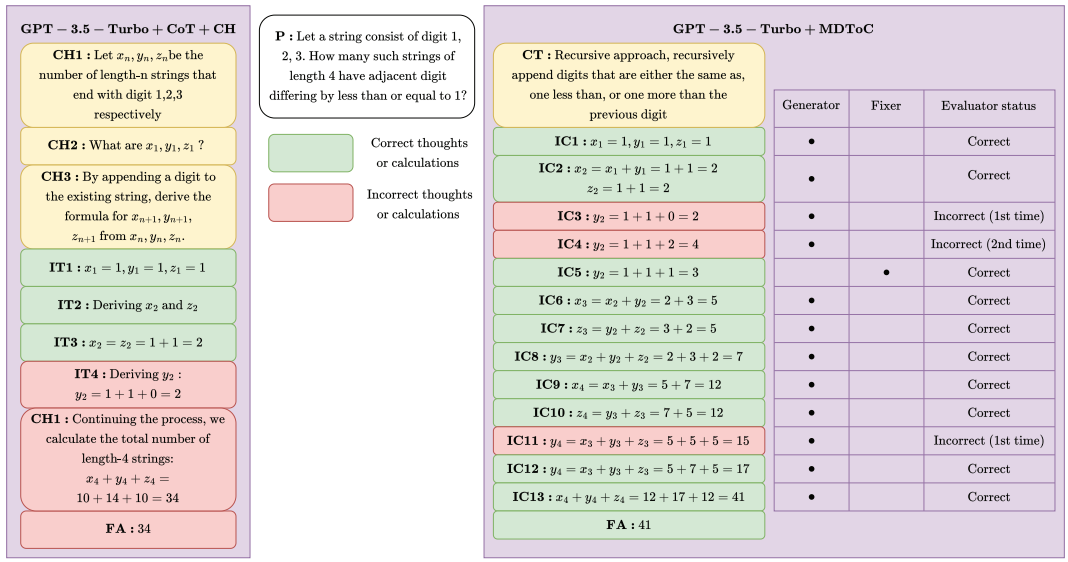


Fig. 5. Comparative analysis of reasoning steps: GPT-3.5-Turbo with CoT and CH versus GPT-3.5-Turbo with our MDToC approach. Subfigure (A) displays GPT-3.5-Turbo's reasoning with CoT, supplemented by annotated concepts and hints **(CH)** intended to guide the model's step-by-step reasoning; **IT** denotes intermediate thoughts, and **FA** indicates the final answer. Although these conceptual hints attempt to structure the problem-solving process, GPT-3.5-Turbo still yields an incorrect count of **34**. Because there is no automatic mechanism to spot and correct mistakes in intermediate steps, the model's calculation errors persist through to the final answer. Subfigure (B) shows our proposed concept tree (CT) approach under a multi-attempt evaluator and fixer framework, referred to here as MDToC. **IC** stands for intermediate calculations, and each is evaluated by an evaluator component. In this example, **IC3** and **IC4** are identified as incorrect, triggering the fixer to regenerate corrected values in **IC5**. This iterative refine-and-fix process avoids propagating calculation errors, ultimately yielding the correct final answer **FA** of **41**. Notably, this process requires no extra annotated hints, only the concept tree plus repeated evaluation up to 2 attempts, a threshold chosen to reduce the risk of model "hallucinations" (erroneous or fabricated steps).

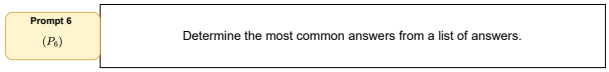


Fig. 6. Prompts for the reviewing phase.

and analysis. Game-of-24 employs $c_s = 15$ and $t = 5$, focusing on broader analysis of calculation combinations.

3.3. Model Configuration

In our experiments, we use OpenAI LLMs throughout. GPT-4o handles the planning phase for concept diversification and the review phase for response standardization and voting. For the monitoring phase's fixer component p_{θ_f} (see Fig. 2), we deploy GPT-4o-mini for cost efficiency while maintaining accuracy. The generator, evaluator, and solver components (see Fig. 2) utilize four different models (GPT-3.5-Turbo, GPT-4o-mini, GPT-4-Turbo, and GPT-4o) to enable direct comparison with other prompting methods. Therefore, GPT-3.5-Turbo+MDToC, GPT-4o-mini+MDToC, GPT-4-Turbo+MDToC, and GPT-4o+MDToC schemes each use their namesake model for monitoring phase components, while all sharing GPT-4o for planning/reviewing and GPT-4o-mini for fixing.

To demonstrate the fairness of these comparisons despite including GPT-4o and GPT-4o-mini, we analyzed token consumption across all models and phases for three datasets in Table 1. The analysis shows that GPT-4o for the planning and reviewing phase uses less than 1% of total tokens for both CHAMP and MATH datasets and approximately 2% for Game-of-24. In the fixer component, GPT-4o-mini consumes 23,428, 21,599, and 1,945 tokens for CHAMP, MATH, and Game-of-24 respectively, accounting for 4.1% to 6.4% of total tokens in the monitoring phase. Since the backbone model is responsible for 91.6% to 95.4% of overall token usage, we can confidently make valid comparisons with alternative prompting techniques that utilize these four primary GPT models.

Table 1. Average tokens used per response of GPT-4o, GPT-3.5-Turbo, GPT-4o-mini, and GPT-4-Turbo across the planning, reviewing, and monitoring phases for the CHAMP, MATH, and Game-of-24 (G24).

GPT	*Phase*	*Dataset*		
		CHAMP	*MATH*	*G24*
4o	Planning	2,671	2,292	655
	Reviewing	346	424	127
3.5-Turbo	Generator	548,202	472,175	36,151
4o-mini	Evaluator	465,420	443,532	34,437
4-Turbo	Solver	433,588	378,745	33,804
4o		367,091	316,275	29,668
4o-mini	Fixer	23,428	21,599	1,945

Table 2. Comparative performance of different prompting approaches for various GPTs on the CHAMP dataset. '0-shot' and '5-shot' denote in-context examples in prompts; '1/3 sln' indicates the proportion of complete solution provided; CoT and CH represent Chain-Of-Thought and Annotated concepts and hints in prompts.

Prompt	*GPT*			
	3.5-Turbo	*4o-mini*	*4-Turbo*	*4o*
0-shot	28.5	36.2	41.9	55.3
CoT	29.6	36.5	37.8	55.2
5-shot	34.8	38.7	43.1	56.5
CoT + CH	34.4	42.3	53.0	60.0
1/3 sln	33.7	43.4	53.7	63.5
ToT	31.7	37.8	52.7	61.3
GoT	30.5	39.6	53.1	60.9
Our MDToC	**39.5**	**48.3**	**58.1**	**68.2**

3.4. CHAMP evaluation

Table 2 compares various prompting techniques across GPT-3.5-Turbo, GPT-4o-mini, GPT-4-Turbo, and GPT-4o models. While traditional approaches like zero-shot, CoT, 5-shot prompting show modest results, incorporating concepts and hints into prompting techniques demonstrates greater success. CoT + CH improves accuracy, with partial solution provision (1/3 sln) offering further gains. For instance, GPT-4-Turbo achieves 53.0% accuracy with CoT+CH, significantly outperforming its 37.8% CoT and 43.1% 5-shot results.

Our MDToC demonstrates superior performance across all models, achieving 39.5% (GPT-3.5-Turbo), 48.3% (GPT-4o-mini), 58.1% (GPT-4-Turbo), and 68.2% (GPT-4o). These results represent substantial improvements over ToT and GoT, with gains ranging from 5.0% to 10.5%, underscoring the effectiveness of dynamically structuring relevant concepts and employing a metacognitive feedback mechanism to refine calculation steps.

Table 3. Comparative performance of different prompting approaches for various GPTs on the MATH dataset

Prompt	*GPT*			
	3.5-Turbo	*4o-mini*	*4-Turbo*	*4o*
0-shot	45.7	71.4	72.6	81.5
CoT	48.6	72.4	73.3	81.6
5-shot	54.3	77.1	79.5	82.4
ToT	53.2	78.5	80.4	87.1
GoT	51.8	80.0	81.2	87.6
Our MDToC	**60.8**	**83.8**	**86.6**	**89.5**

3.5. MATH evaluation

In Table 3, 0-shot prompts yield 45.7% for GPT-3.5-Turbo, 71.4% for GPT-4o-mini, 72.6% for GPT-4-Turbo, and 81.5% for GPT-4o on the MATH dataset. Adding reasoning steps through CoT improves these scores slightly (e.g., from 45.7% to 48.6% for GPT-3.5-Turbo), while increasing the number of examples (5-shot) provides further gains (up to 79.5% for GPT-4-Turbo and 82.4% for GPT-4o). The ToT and GoT methods surpass standard few-shot prompts for the two more advanced models, pushing GPT-4-Turbo to 80.4% and 81.2% and GPT-4o to 87.1% and 87.6%, a notable jump of around 7% from CoT.

Even so, our MDToC outperforms all these strategies, achieving 60.8% for GPT-3.5-Turbo, 83.8% for GPT-4o-mini, 86.6% for GPT-4-Turbo, and 89.5% for GPT-4o. Compared to ToT, our MDToC provides an extra 7.6% boost for GPT-3.5-Turbo, 5.3% for GPT-4o-mini, and 6.2% for GPT-4-Turbo. These results strengthen our claim that our MDToC not only overcomes the evaluation constraints in ToT but also achieves more robust performance than purely tree-based approaches like ToT.

3.6. Game-of-24 evaluation

Table 4 reports six prompting strategies (0-shot, CoT, 5-shot, ToT, GoT, and our MDToC) evaluated on four models (GPT-3.5-Turbo, GPT-4o-mini, GPT-4-Turbo, and the GPT-4o). The broad pattern remains: minimal prompting (0-shot or CoT) yields very low accuracy (2-10%), while adding a handful of demonstrations (5-shot) produces a modest gain (6-18%). ToT then brings a significant jump for GPT-4o-mini to 56%, GPT-4-Turbo to 74%, and GPT-4o to 88%. GoT raises scores further to 62%, 81%, and 90% for GPT-4o-mini, GPT-4-Turbo,

Table 4. Comparative performance of different prompting approaches for various GPTs on the Game-of-24 dataset.

Prompt	*GPT*			
	3.5-Turbo	*4o-mini*	*4-Turbo*	*4o*
0-shot	2	3	4	10
CoT	3	3	4	9
5-shot	6	8	10	18
ToT	19	56	74	88
GoT	21	62	81	90
Our MDToC	**30**	**75**	**85**	**90**

Table 5. Per-case cost and accuracy for our MDToC with GPT models on MATH, CHAMP, and Game-of-24 (G24). W stands for using GPT-4o-mini in the fixer component and GPT-4o in the planning and reviewing phase in our MDToC configuration. W/o stands for using the same LLMs across all components.

Dataset	*GPT*	*Cost ($)*		*Accuracy*	
		w	*w/o*	*w*	*67*
MATH	3.5-Turbo	0.58	0.62	**60.8**	54.1
	4-Turbo	19.79	21.16	86.6	**86.9**
	4o-mini	0.23	0.23	**83.8**	82.9
	4o	3.5	3.7	89.5	**89.7**
CHAMP	3.5-Turbo	0.69	0.72	**39.5**	33.9
	4-Turbo	22.59	24.15	58.1	**59.0**
	4o-mini	0.24	0.24	**48.3**	47.1
	4o	3.94	4.08	68.2	**68.5**
G24	3.5-Turbo	0.03	0.05	**30**	25
	4-Turbo	1.79	1.92	**85**	85
	4o-mini	0.02	0.02	**75**	74
	4o	0.37	0.39	**90**	90

and GPT-4o, respectively; even GPT-3.5-Turbo climbs from 19% (ToT) to 21% (GoT).

Despite these gains, our MDToC still delivers the best accuracy across the board: 30% (+9% over GoT) on GPT-3.5-Turbo, 75% (+13%) on GPT-4o-mini, 85% (+4%) on GPT-4-Turbo, and a top-tied 90% on GPT-4o. These margins underscore the critical role of MDToC's evaluator p_{θ_e} and fixer p_{θ_f}, which detect and repair erroneous intermediate expressions involving the four numbers in Game-of-24, maintaining logical consistency and reducing hallucinations throughout the reasoning chain.

3.7. Cost analysis of our MDToC configuration

Replacing the planner and reviewer with GPT-4o and the fixer with GPT-4o-mini reduces or maintains costs while maintaining or improving accuracy across all backbone models and datasets tested (shown in Table 5). For GPT-3.5-Turbo, costs decreased while accuracy increased on MATH (54.1% to 60.8%), CHAMP (33.9% to 39.5%), and Game-of-24 (25% to 30%). GPT-4-Turbo saw cost reductions with minimal accuracy changes on all datasets, while GPT-4o as backbone showed 3-5% cost savings with negligible accuracy differences ($\leq$0.3%). These results demonstrate that delegating auxiliary roles to lighter models is an effective strategy for reducing computational expenses without compromising performance, with weaker models particularly benefiting from the diverse planning concepts provided by GPT-4o.

Table 6. Hyperparameters (see Sections 2.4.1 and 2.4.2) in MDToC with GPT-4o on MATH and Game-of-24 (G24). Acc stands for Accuracy.

(C_{min}, C_{max}, SC_{min}, SC_{max})	*(c_s, t)*	*MATH Acc*	*G24 Acc*	*MATH Cost*	*G24 Cost*
(2,4,1,2)	(2,10)	**89.5%**	87%	**$3.5**	$0.7
(3,5,1,2)	(2, 10)	89.6%	88%	$4.9	$0.8
(3,5,2,4)	(3, 15)	89.9%	88%	$5.8	$1.0
(1,1,1,1)	(15, 5)	83.4%	**90%**	$2.6	**$0.4**

3.8. Hyperparameter Sensitivity Test

Table 6 shows important hyperparameter combinations. Increasing concept exploration to 5 concepts ($C_{max} = 5$) and up to 20 sub-concepts in total ($SC_{max} = 4$) only marginally improves the accuracy of our MDToC on MATH (by 0.4%). This confirms that about 4 concepts ($C_{max} = 4$) are sufficient. In addition, the cost is significantly lower ($3.5 versus $5.8). For Game-of-24, accuracy slightly decreases (from 90% to 88%) when computational exploration is narrowed in favor of concept variety, indicating that this dataset does not benefit from diverse concepts. Meanwhile, despite the cost decrease (down to $2.6), limiting concepts while broadening computational exploration reduces MATH accuracy (to 83.4%). However, such settings suit exploration-based datasets like Game-of-24.

3.9. Hyperparameter Selection Heuristics

Table 6 suggests a three-step procedure for selecting the parameters of the dynamic tree on a new dataset without exhaustive trial-and-error.

Step 1: Determine whether the dataset benefits from concept exploration or

computational exploration. To do so, inspect the annotated concepts and solutions of a small number of problems. Datasets whose problems require different mathematical concepts (CHAMP, MATH) benefit from concept exploration. In contrast, exploration-based datasets whose problems require diverse calculation combinations (Game-of-24) benefit from computational exploration. This choice has the largest effect in Table 6: removing concept exploration lowers MATH accuracy by 6.1% (89.5% to 83.4%), whereas it raises Game-of-24 accuracy by 3% (87% to 90%).

Step 2: Set the parameters according to Step 1. For concept exploration, use the concept range, sub-concept range, c_s, and t of Section 3.2. Wider ranges are not worthwhile: Table 6 shows that they improve MATH accuracy by at most 0.4% at 40% to 66% higher cost. For computational exploration, set all four tree parameters to 1. Then, use a larger c_s with fewer iterations t (Section 3.2). This setting gives the highest Game-of-24 accuracy (90%) at the lowest cost ($0.4) in Table 6.

Step 3: Confirm the parameters on a small development set of, for example, 20 to 30 held-out problems. Increase C_{max} and SC_{max} by one at a time. Stop when the accuracy gain per additional cost falls below a preset threshold, for example 1% accuracy per additional dollar of per-case cost. In Table 6, the first increment (from $C_{max} = 4$ to 5) gains only 0.1% for $1.4 more per case. This is far below such a threshold. Therefore, the confirmation stops after one increment.

4. Conclusions

Our proposed MDToC framework enhances mathematical reasoning in LLMs through the metacognitive stages of planning, monitoring, and reviewing. It outperforms ToT and GoT techniques, achieving up to 19% higher accuracy than ToT on Game-of-24 and showing consistent improvements on CHAMP and MATH. Our MDToC excels in calculation-intensive tasks through dynamic concept structuring and iterative error correction, establishing a foundation for future research in complex problem-solving.

5. Study limitations

Despite MDToC's superior performance compared to previous prompting methodologies, several notable limitations affect its practical implementation. First, our metacognitive calculation approach exhibits domain-specific constraints, particularly in mathematical fields such as geometry, where spatial reasoning predominates over calculation verification. In such domains, the iterative verification processes central to MDToC may offer diminishing returns, as the primary cognitive challenges relate to geometry visualization rather than computational validation. Two mitigations could extend MDToC to such domains. First, the generator and evaluator components could use multimodal LLMs, so that each step is evaluated against the figure itself. Second, an explicit spatial parsing tool could translate geometric figures into symbolic representations such as coordinates and angle relations, as in AlphaGeometry [32]. These representations would make spatial reasoning steps evaluable in the same way as calculations. We leave this to future work.

Second, the performance improvements delivered by our MDToC incur computational costs. As demonstrated in our analyses (see Tables 1 and 5), the method introduces significant resource overhead, particularly when implementing MDToC with expensive LLMs such as GPT-4-Turbo. Token consumption for CHAMP and MATH benchmark problems reaches approximately $450{,}000$ tokens per problem, translating to approximately $\$20$ per problem, a cost scale that may prove prohibitive for educational institutions and research organizations with limited budgets. These economic constraints potentially restrict MDToC's deployment in resource-limited environments.

Third, the best parameters of the dynamic tree differ between datasets that benefit from concept exploration and exploration-based datasets (Table 6). Section 3.9 provides a selection procedure, but a small development set is still needed to confirm the parameters on a new dataset. In addition, the dynamic tree of concepts requires the backbone LLM to follow the dependencies among concepts, sub-concepts, and calculations. Such deep relational reasoning remains a limitation even for strong LLMs such as DeepSeek-R1 [33].

CRediT author statement

- **Tung Duong Ta:** Conceptualization, Methodology, Software, Validation, Investigation, Data curation, Writing original draft, Visualization.
- **Tim Oates:** Writing - review and editing.
- **Thien Van Luong:** Writing - review and editing.
- **Huan Vu:** Writing - review and editing.
- **Tien Cuong Nguyen:** Writing - review and editing.

COMPETING INTERESTS

The authors declare that there is no conflict of interest regarding the publication of this article.

REFERENCES

[1] J. Achiam, S. Adler, S. Agarwal, et al. (2023), "GPT-4 technical report", *arXiv*, DOI: 10.48550/arXiv.2303.08774.

[2] Anthropic (2024), "The Claude 3 model family: Opus, Sonnet, Haiku", *Claude-3 Model Card*, **1**, pp.1-42.

[3] K. Cobbe, V. Kosaraju, M. Bavarian, et al. (2021), "Training verifiers to solve math word problems", *arXiv*, DOI: 10.48550/arXiv.2110.14168.

[4] A. Patel, S. Bhattamishra, N. Goyal (2021), "Are NLP models really able to solve simple math word problems?", *arXiv*, DOI: 10.48550/arXiv.2103.07191.

[5] D. Hendrycks, C. Burns, S. Kadavath, et al. (2021), "Measuring mathematical problem solving with the MATH dataset", *arXiv*, DOI: 10.48550/arXiv.2103.03874.

[6] Y. Mao, Y. Kim, Y. Zhou (2024), "CHAMP: A Competition-level Dataset for Fine-Grained Analyses of LLMs' Mathematical Reasoning Capabilities", *arXiv*, DOI: 10.48550/arXiv.2401.06961.

[7] N. Patel, M. Kulkarni, M. Parmar, et al. (2024), "Multi-LogiEval: Towards Evaluating Multi-Step Logical Reasoning Ability of Large Language Models", *arXiv*, DOI: 10.48550/arXiv.2406.17169.

[8] N. Tyagi, M. Parmar, M. Kulkarni, et al. (2024), "Step-by-Step Reasoning to Solve Grid Puzzles: Where do LLMs Falter?", *arXiv*, DOI: 10.48550/arXiv.2407.14790.

[9] P.Y. Wang, T.S. Liu, C. Wang, et al. (2026), "A survey on large language models for mathematical reasoning", *ACM Computing Surveys*, **58(8)**, pp.1-35.

[10] J. Wei, X. Wang, D. Schuurmans, et al. (2022), "Chain-of-thought prompting elicits reasoning in large language models", *Advances in Neural Information Processing systems*, **35**, pp.24824-24837.

[11] X. Wang, J. Wei, D. Schuurmans, et al. (2023), "Self-consistency improves chain of thought reasoning in language models", *arXiv*, DOI: 10.48550/arXiv.2203.11171.

[12] S. Yao, D. Yu, J. Zhao, et al. (2024), "Tree of thoughts: Deliberate problem solving with large language models", *Advances in Neural Information Processing Systems*, **36**, DOI: 10.48550/arXiu.2305.10601.

[13] J. Long (2023), "Large language model guided tree-of-thought", *arXiv*, DOI: 10.48550/arXiv.2305.08291.

[14] M. Besta, N. Blach, A. Kubicek, et al. (2024), "Graph of thoughts: Solving elaborate problems with large language models", *Proceedings of the AAAI Conference on Artificial Intelligence*, **38**, pp.17682-17690.

[15] Y. Yao, Z. Li, H. Zhao (2023), "Beyond chain-of-thought, effective graph-of-thought reasoning in language models", *arXiv*, DOI: 10.48550/arXiv.2305.16582.

[16] O. Fagbohun, R.M. Harrison, A. Dereventsov (2024), "An empirical categorization of prompting techniques for large language models: A Practitioner's Guide", *arXiv*, DOI: 10.48550/arXiv.2402.14837.

[17] E.R. Lai (2011), "Metacognition: A literature review", *Always learning: Pearson Research Report*, **24**, pp.1-40.

[18] Y. Wang, Y. Zhao (2023), "Metacognitive prompting improves understanding in large language models", *arXiv*, DOI: 10.48550/arXiv.2308.05342.

[19] T. Kojima, S.S. Gu, M. Reid, et al. (2022), "Large language models are zero-shot reasoners", *Advances in Neural Information Processing Systems*, **35**, pp.22199-22213.

[20] T. Brown, B. Mann, N. Ryder, et al. (2020), "Language models are few-shot learners", *Advances in Neural Information Processing Systems*, **33**, pp.1877-1901.

[21] Y. Zhou, Z. Liu, J. Jin, et al. (2024), "Metacognitive retrieval-augmented large language models", *Proceedings of the ACM on Web Conference 2024*, pp.1453-1463.

[22] A. Didolkar, A. Goyal, N.R. Ke, et al. (2024), "Metacognitive capabilities of LLMs: An exploration in mathematical problem Solving", *arXiv*, DOI: 10.48550/arXiv.2405.12205.

[23] X. Chen, R. Aksitov, U. Alon, et al. (2023), "Universal self-consistency for large language model generation", *arXiv*, DOI: 10.48550/arXiv.2311.17311.

[24] A. Madaan, N. Tandon, P. Gupta, et al. (2023), "Self-refine: Iterative refinement with self-feedback", *Advances in neural Information Processing Systems*, **36**, pp.46534-46594.

[25] Y. Li (2025), "Decomposing LLM self-correction: The accuracy-correction paradox and error depth hypothesis", *arXiv*, DOI: 10.48550/arXiv.2601.00828.

[26] Q. Long, K.J. Jiang, J. Chen, et al. (2026), "Self-verification dilemma: experience-driven suppression of Overused Checking in LLM reasoning", *arXiv*, DOI: 10.48550/arXiv.2602.03485.

[27] Z. Shao, P. Wang, Q. Zhu, et al. (2024), "DeepSeekMath: Pushing the limits of mathematical reasoning in open language models", *arXiv*, DOI: 10.48550/arXiv.2402.03300.

[28] J. Zhang, G. Ma, S. Liu, et al. (2026), "STRIDE: Learnable stepwise language feedback for LLM Reasoning", *arXiv*, DOI: 10.48550/arXiv.2605.18851.

[29] K.Y.L. Ku, I.T. Ho (2010), "Metacognitive strategies that enhance critical thinking", *Metacognition and Learning*, **5(3)**, pp.251-267.

[30] AoPS (2025), "AIME Problems", *Art of Problem Solving*, `https://artofproblemsolving.com/community/c3416_aime_problems`, accessed 10 January 2025.

[31] AoPS (2025), "AMC 10 Math Problems", *Art of Problem Solving*, `https://artofproblemsolving.com/community/c3414_amc_10`, accessed 10 January 2025.

[32] T.H. Trinh, Y. Wu, Q.V. Le, et al. (2024), "Solving olympiad geometry without human demonstrations", *Nature*, **625(7995)**, pp.476-482.

[33] C.C. So, Y. Sun, J.M. Wang, et al. (2025), "Are large language models capable of deep relational reasoning? insights from DeepSeek-R1 and benchmark comparisons", *2025 IEEE International Conference on Artificial Intelligence Testing (AITest)*, pp.168-177.