MDToC: Metacognitive Dynamic Tree of Concepts for Boosting Mathematical Problem-Solving of Large Language Models

Anonymous ACL submission

Abstract

Despite advances in mathematical reasoning capabilities, Large Language Models (LLMs) still struggle with calculation verification when using established prompting techniques. We present MDToC (Metacognitive Dynamic Tree of Concepts), a three-phase approach that 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 surpasses existing prompting methods across all backbone models, yielding improvements of up to 7.6% over ToT and 6.2% over GoT, establishing metacognitive calculation verification as a promising direction for enhanced mathematical reasoning.

1 Introduction

Large Language Models (LLMs) like GPT-4 (Achiam et al., 2023) and Claude (Anthropic, 2024) demonstrate proficiency in various mathematical problems, excelling in easy to medium-difficulty tasks as evidenced by their performance on benchmarks such as GSM8k (Cobbe et al., 2021) and SVAMP (Patel et al., 2021). However, their efficacy diminishes when faced with complex challenges presented in datasets such as MATH (Hendrycks et al., 2021) and CHAMP (Mao et al., 2024). 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 (Patel et al., 2024; Tyagi et al., 2024). 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.

Researchers have widely adopted prompting techniques, particularly Chain-of-Thought (CoT) (Wei et al., 2022) and self-consistency CoT (SC-CoT) (Wang et al., 2023), 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 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 (Yao et al., 2024). Therefore, it is essential to develop more robust reasoning methodologies.

Recent hierarchical prompting techniques like Tree-of-Thoughts (ToT) (Yao et al., 2024) (Long, 2023) and Graph-of-Thoughts (GoT) (Besta et al., 2024) (Yao et al., 2023) have advanced reasoning capabilities through structured thought representation and intermediate evaluation, achieving impressive results on complex tasks (74% accuracy on Game-of-24 (Yao et al., 2024) and 89% on Sequence-Sorting-64-elements (Besta et al., 2024), respectively). However, as shown in Figure 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 (Fagbohun et al., 2024). In the field of psychology, metacognition enables individuals to reflect on and critically analyze their thought processes (Lai, 2011). Recent research has enhanced model capabilities with metacognitive processes for natural language understanding tasks. For example, (Wang and Zhao, 2023) demonstrated that LLMs prompted with metacognitive thinking outperformed previous techniques such as zero-shot (Kojima et al., 2022) (Brown et al., 2020) or CoT prompting (Wei et al., 2022) across various NLP tasks. (Zhou et al., 2024) 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 (Didolkar et al., 2024), 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.

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 (Chen et al., 2023). 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%).

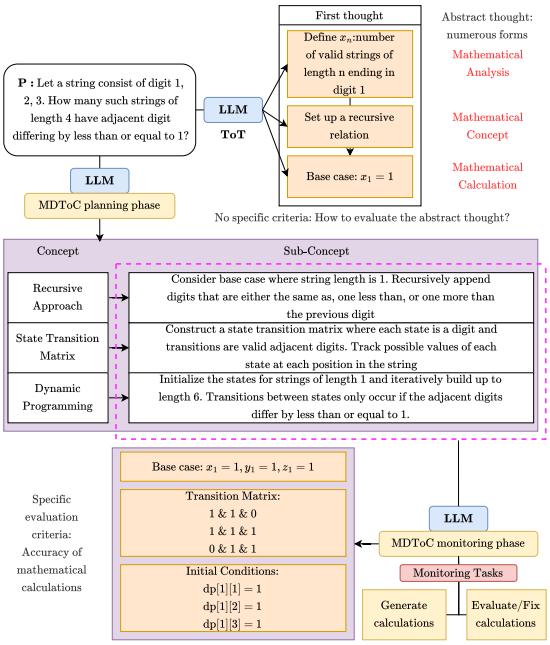

Figure 1: ToT prompting yields initial abstract thoughts (e.g., analyses, concepts, calculations; in red), which are challenging to evaluate due to the intangible nature of conceptual reasoning and the lack of specific criteria to measure their correctness or completeness. Our MDToC addresses these abstract 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.

2 Related Work

2.1 Prompting techniques

Let p_θ be a LLM parameterized by θ. Two 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 (Wei et al., 2022). 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 (Yao et al., 2024). The

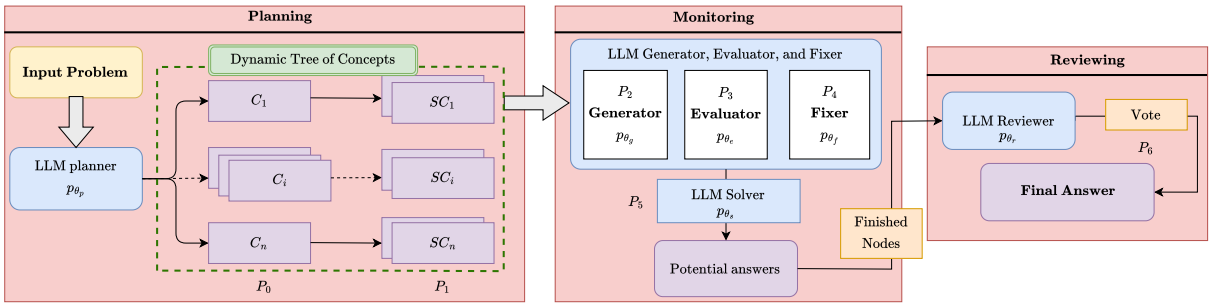

Figure 2: Proposed MDToC prompting structure. C represents the first-depth concept, while SC represents the second-depth sub-concept. P_0 and P_1 are prompts used in the planning phase shown in Figure 3, while P_2, P_3, P_4, and P_5 are prompts used in the monitoring phase given in Figure 4. Prompts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

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 (Besta et al., 2024). 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 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 (Ku and Ho, 2010), 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.

3 Methodology

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. Figure 2 shows a visual representation of our methodology.

3.1 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 $V = \{c_1^{d=1}, c_2^{d=1}, ..., c_n^{d=1}\}$.

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. Figure 3 demonstrates our prompts for P_0 and P_1. Our tree T is then expressed as:

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

.

Prompt 0 (P_0)	Generate a minimum of C_min and a maximum of C_max distinct problem-specific concepts
Prompt 1 (P_1)	Extract all problem-solving information. Then generate a minimum of SC_min and a maximum of SC_max distinct sub-concepts filled with the extracted information given an explored concept

Figure 3: Prompts for the planning phase.

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 Figure 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 (Mao et al., 2024) 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.

Prompt 2 (P_2)	Derive the next small calculation step to deduce the partial solution. Any unknown computation not in the solution is not allowed for the deduction. Do not generate explanation texts of the step.
Prompt 3 (P_3)	Verify if the next calculation is correctly computed. Use code to confirm your verification. Only verify the accuracy.
Prompt 4 (P_4)	Recalculate the next calculation step-by-step. Verify if the output of the recalculation matches the original calculation. If not match, fix the output!
Prompt 5 (P_5)	Given the partial solution, continue solving and derive the final answer!

Figure 4: Prompts for the monitoring phase.

3.2 Monitoring

In the monitoring phase, we employ a ToT-based structure (Yao et al., 2024) 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 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 calcula-

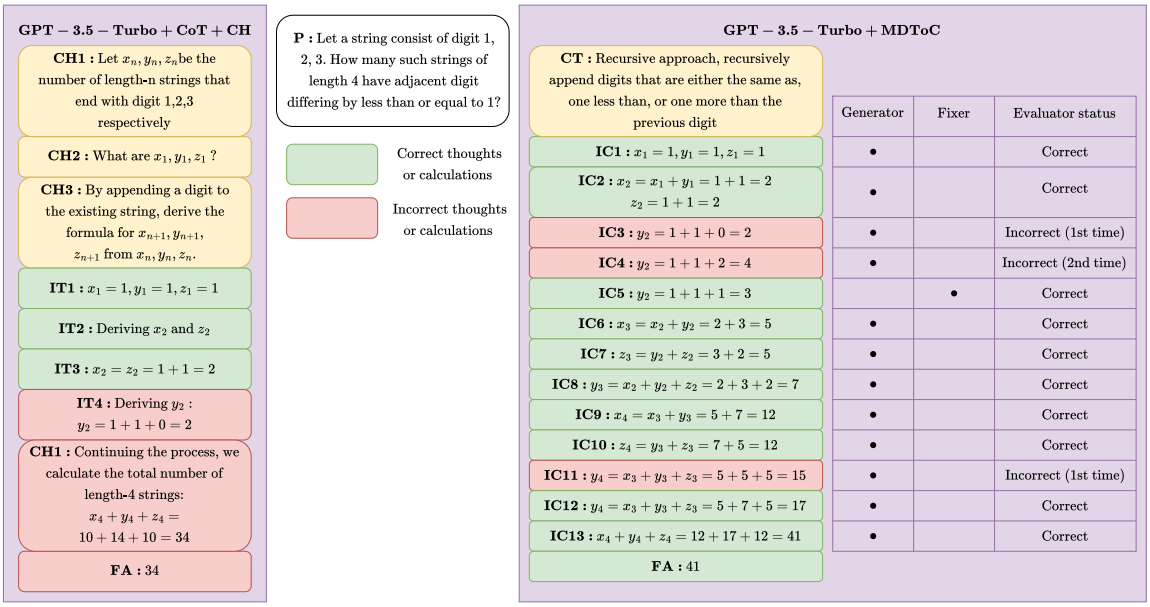

Figure 5: **Comparative analysis of reasoning steps: GPT-3.5-Turbo with CoT and CH versus GPT-3.5-Turbo with our MDToC approach**. Subfigure (a) displays GPT-3.5-Turbo's reasoning with CoT, supplemented by annotated concepts and hints **(CH)** intended to guide the model's step-by-step reasoning; **IT** 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–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).

tions 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} 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. Figure 4 illustrates our prompts P_2, P_3, P_4, and P_5.

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

3.2.1 Example Analysis

Figure 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. Figure 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, Figure 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.

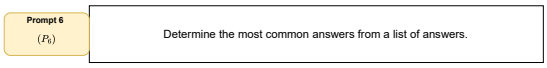

Figure 6: Prompts for the reviewing phase.

3.3 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 Figure 6 to conduct a majority voting of A. This stage involves finding the most common answer from A, returning the most common one as $\tilde{a} \sim p_{\theta_e}(A)$, where $\tilde{a}$ is the final solution to the objective q.

4 Experiments

4.1 Dataset

Our MDToC approach is evaluated using three datasets: CHAMP (Mao et al., 2024) (270 high school-level competition math problems across five categories, providing insight into concept tree-driven reasoning), MATH (Hendrycks et al., 2021) (12,500 competition-level problems from sources like AIME (AoPS, a) and AMC 10/12 (AoPS, b) 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) (Yao et al., 2024).

4.2 Parameters

Planning phase For the CHAMP and MATH datasets, we set $C_{min} = 2$, $C_{max} = 3$, $SC_{min} = 1$, and $SC_{max} = 2$ (see our 3.1). 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 our 3.2), emphasizing in-depth concept decomposition and analysis. Game-of-24 employs $c_s = 10$ and $t = 4$, focusing on broader analysis of calculation combinations.

4.3 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 about 6% of total tokens in the monitoring phase. Since the remaining three GPT models are responsible for roughly 93% 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 Evaluator Solver	548,202	472,175	36,151
4o-mini		465,420	443,532	34,437
4-Turbo		433,588	378,745	33,804
4o		367,091	316,275	29,668
4o-mini	Fixer	23,428	21,599	1,945

4.4 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, five-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.

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' indicate 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**

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 65.2% (GPT-4o). These results represent substantial improvements over ToT and GoT, with gains ranging from 5.4% to 10.5%, underscoring the effectiveness of dynamically structuring relevant concepts and employing a metacognitive feedback mechanism to refine calculation steps.

4.5 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 79.3% for GPT-4-Turbo and 87.1% for GPT-4o). The ToT and GoT method 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.

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**

4.6 Game-of-24 evaluation

Table 4 now 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 new 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, 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.

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**

5 Discussion

5.1 Cost analysis of our MDToC configuration

Dataset	GPT	Cost ($)		Accuracy	
		w	w/o	w	w/o
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

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.

Replacing the planner and reviewer with GPT-4o and the fixer with GPT-4o-mini reduces 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 5-7% 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.

5.2 Hyperparameter Sensitivity Test

$(C_{min}, C_{max}, SC_{min}, SC_{max})$	(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**

Table 6: Hyperparameters (see our 3.1 and 3.2) in MDToC with GPT-4o on MATH and Game-of-24 (G24). Acc stands for Accuracy.

Table 6 shows important hyperparameter combinations. Increasing concept exploration (up to 5 concepts ($C_{\text{max}} = 5$ and 20 subconcepts in total ($SC_{\text{max}} = 4$) marginally improves the accuracy of our MDToC on MATH (0.4%), confirming our finding that about 4 concepts ($C_{\text{max}} = 4$) subconcepts are sufficient while reducing the cost significantly (from $5.8 to $3.5). For Game-of-24, the accuracy slightly decreases (90% to 88%) with less broad explorations, indicating this dataset does not benefit from concept variety. Meanwhile, despite of cost decrease (down to $2.6), limited concepts with broad computational exploration reduce MATH accuracy (to 83.4%). However, these suit exploration-based datasets like Game-of-24.

6 Conclusion

Our proposed MDToC framework enhances mathematical reasoning in LLMs through structured metacognition—planning, monitoring, and reviewing. It outperforms ToT and GoT techniques, achieving up to 11% higher accuracy 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.

7 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.

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.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

AI Anthropic. 2024. The claude 3 model family: Opus, sonnet, haiku. *Claude-3 Model Card*, 1.

AoPS. a. Aime problems. https://artofproblemsolving.com/community/c3416_aime_problems. Accessed: 2025-01-10.

AoPS. b. Amc 10 math problems. https://artofproblemsolving.com/community/c3414_amc_10. Accessed: 2025-01-10.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and 1 others. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17682–17690.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Xinyun Chen, Renat Aksitov, Uri Alon, Jie Ren, Kefan Xiao, Pengcheng Yin, Sushant Prakash, Charles Sutton, Xuezhi Wang, and Denny Zhou. 2023. Universal self-consistency for large language model generation. *arXiv preprint arXiv:2311.17311*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems, 2021. *URL https://arxiv.org/abs/2110.14168*.

Aniket Didolkar, Anirudh Goyal, Nan Rosemary Ke, Siyuan Guo, Michal Valko, Timothy Lillicrap, Danilo Rezende, Yoshua Bengio, Michael Mozer, and Sanjeev Arora. 2024. Metacognitive capabilities of llms: An exploration in mathematical problem solving. *arXiv preprint arXiv:2405.12205*.

Oluwole Fagbohun, Rachel M Harrison, and Anton Dereventsov. 2024. An empirical categorization of prompting techniques for large language models: A practitioner's guide. *arXiv preprint arXiv:2402.14837*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

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

KYL Ku and IT Ho. 2010. Metacognitive strategies that enhance critical thinking. metacognition and learning, 5 (3), 251–267.

Emily R Lai. 2011. Metacognition: A literature review. *Always learning: Pearson research report*, 24:1–40.

Jieyi Long. 2023. Large language model guided tree-of-thought. *arXiv preprint arXiv:2305.08291*.

Yujun Mao, Yoon Kim, and Yilun Zhou. 2024. Champ: A competition-level dataset for fine-grained analyses of llms' mathematical reasoning capabilities. *arXiv preprint arXiv:2401.06961*.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are nlp models really able to solve simple math word problems? *arXiv preprint arXiv:2103.07191*.

Nisarg Patel, Mohith Kulkarni, Mihir Parmar, Aashna Budhiraja, Mutsumi Nakamura, Neeraj Varshney, and Chitta Baral. 2024. Multi-logieval: Towards evaluating multi-step logical reasoning ability of large language models. *arXiv preprint arXiv:2406.17169*.

Nemika Tyagi, Mihir Parmar, Mohith Kulkarni, Aswin RRV, Nisarg Patel, Mutsumi Nakamura, Arindam Mitra, and Chitta Baral. 2024. Step-by-step reasoning to solve grid puzzles: Where do llms falter? *arXiv preprint arXiv:2407.14790*.

X Wang, J Wei, D Schuurmans, Q Le, E Chi, S Narang, A Chowdhery, and D Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. arxiv. *arXiv preprint arXiv:2203.11171*.

Yuqing Wang and Yun Zhao. 2023. Metacognitive prompting improves understanding in large language models. *arXiv preprint arXiv:2308.05342*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

Yao Yao, Zuchao Li, and Hai Zhao. 2023. Beyond chain-of-thought, effective graph-of-thought reasoning in language models. *arXiv preprint arXiv:2305.16582*.

Yujia Zhou, Zheng Liu, Jiajie Jin, Jian-Yun Nie, and Zhicheng Dou. 2024. Metacognitive retrieval-augmented large language models. In *Proceedings of the ACM on Web Conference 2024*, pages 1453–1463.

A Appendix

A.1 Complexity Analysis

Table 7 demonstrates that our MDToC achieves higher accuracy than ToT. On the MATH benchmark with GPT-4-Turbo, our MDToC with GPT-4-Turbo raises accuracy from 80.4% to 86.6% (+6.2 % gain) while consuming only 16% more tokens (346k → 403k) and about 40 seconds of extra time — yielding roughly 1% of extra accuracy for every 2.5% of extra tokens. A similar pattern appears on CHAMP: our MDToC with GPT-4-Turbo delivers 5.4% gain (52.7% → 58.1%) for just 14% more tokens and a 0.2-minute latency increase. On the Game-of-24 task, our MDToC converts a five-fold token increase into an 11% accuracy jump (74% → 85%) for GPT-4-Turbo while keeping runtime under 2 minutes. With GPT-4o, for example, our MDToC converts a 28% rise in tokens on MATH into 2.4% accuracy gain over ToT (87.1% → 89.5%) and turns a 26% token increase on CHAMP into 6.9% accuracy gain (61.3% → 68.2%). These results confirm that the added metacognitive steps for calculation evaluation pay for themselves: each additional cost generates more correct answers than ToT can perform with the same model.

Dataset	Prompt	GPT	Token	Cost	Time	Acc
MATH	MDToC	4-Turbo	403,060	19.79	12.1	**86.6**
		4o	286,935	3.5	11.1	**89.5**
	ToT	4-Turbo	**346,193**	**12.98**	**11.5**	80.4
		4o	**223,088**	**2.79**	**11.0**	87.1
CHAMP	MDToC	4-Turbo	460,033	22.59	12.3	**58.1**
		4o	315,561	3.94	11.4	**68.2**
	ToT	4-Turbo	**404,881**	**15.18**	**12.1**	52.7
		4o	**249,713**	**3.12**	**11.1**	61.3
G24	MDToC	4-Turbo	36,531	1.79	1.8	**85**
		4o	29,464	0.37	1.7	**90**
	ToT	4-Turbo	**6,958**	**0.74**	**1.0**	74
		4o	**6,179**	**0.08**	**1.0**	88

Table 7: Token and cost analysis of our MDToC and ToT on MATH, CHAMP, and Game-of-24 (G24). The token and cost are **per case**. Time is measured in **minutes**. Acc stands for Accuracy.

A.2 LLM Benchmark

To better understand the performance of our MDToC, Table 8 shows the performance of our MDToC when using the same GPT models and other open-source LLMs across all three components. Across a broad sweep of language models, our MDToC consistently outperforms both ToT and GoT. With GPT-4-Turbo, accuracy rises from 52.7% (ToT) and 54.2% (GoT) to 59.0% on CHAMP, from 80.4% and 80.8% to 86.9% on MATH, and from 79% and 81% to 85% on Game-24. Notably, the gains are even larger for smaller models: GPT-4o-mini sees a 9.3-point jump on CHAMP and an 8.9-point boost on Game-24, underscoring MDToC's ability to compensate for limited parameter capacity.

The trend extends to open-source LLMs. When applied to Mistral-7B, our MDToC improved MATH accuracy from 23.5% to 27.2% (+3.7% increase), CHAMP accuracy from 18.1% to 19.7%, and Game-of-24 accuracy from 8% to 9%. For the stronger Mistral 8×22B model, MDToC yielded improvements on MATH (52.7% to 55.4%), CHAMP (31.9% to 34.5%), and Game-of-24 (22% to 26%). The Llama-3 family showed similar benefits: the 8B variant experienced increases on MATH (26.8%

LLM	Prompt	Dataset		
		CHAMP	MATH	G24
GPT-3.5-Turbo	ToT	31.7	51.2	19
	GoT	32.3	51.6	21
	MDToC	**33.9**	**54.1**	**25**
GPT-4-Turbo	ToT	52.7	80.4	79
	GoT	54.2	80.9	81
	MDToC	**59.0**	**86.9**	**85**
GPT-4o-mini	ToT	37.8	78.5	56
	GoT	39.3	79.0	62
	MDToC	**47.1**	**82.9**	**74**
GPT-4o	ToT	61.3	87.1	88
	GoT	63.4	87.9	90
	MDToC	**68.5**	**89.7**	**90**
Mistral-7B	ToT	18.1	23.5	8
	GoT	18.8	24.2	9
	MDToC	**19.7**	**27.2**	**9**
Mistral 8x22B	ToT	31.9	52.7	22
	GoT	32.4	53.2	23
	MDToC	**34.5**	**55.4**	**26**
Llama-3 8B	ToT	19.2	26.8	7
	GoT	20.9	27.5	8
	MDToC	**21.5**	**30.6**	**10**
Llama-3 70B	ToT	31.3	52.1	24
	GoT	32.7	53.6	26
	MDToC	**36.1**	**57.3**	**29**

Table 8: Accuracy achieved by MDToC, ToT, and GoT on the CHAMP, MATH, and Game-24 datasets using eight language models: GPT-3.5-Turbo, GPT-4o-mini, GPT-4-Turbo, GPT-4o, Mistral-7B, Mistral-8×22B, Llama-3-8B, and Llama-3-70B

to 30.6%) and CHAMP (19.2% to 21.5%), while the 70B variant saw substantial gains on MATH (52.1% to 57.3%), CHAMP (31.3% to 36.1%), and Game-of-24 (24% to 29%). These consistent performance improvements of 2-5% across four open-source LLMs demonstrate that our MDToC enhances community models, mirroring the benefits previously observed with GPT architectures.

A.3 Problem types on MATH dataset

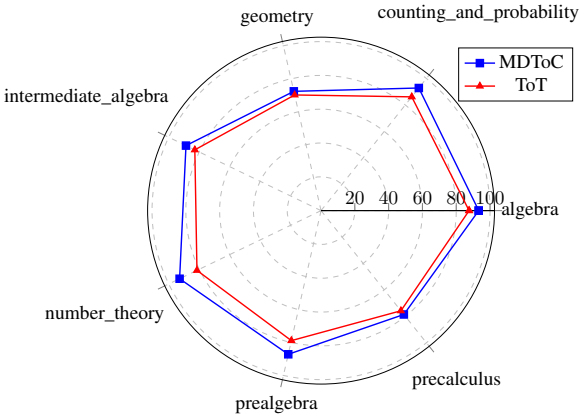

Figure 7: **Radar chart comparing our MDToC and ToT performance on the MATH dataset** in terms of the percentage accuracy—both evaluated with GPT-4-Turbo—on various math problems of this dataset.

Figure 7 employs GPT-4-Turbo—selected for its superior performance over GPT-3.5-Turbo and GPT-4o-mini—to evaluate two prompting methods, ToT and our MDToC. Results were collected across seven math topics (algebra, counting and probability, geometry, intermediate algebra, number theory, prealgebra, and precalculus). MDToC outperformed ToT in five categories, showing notable leads in algebra (93.3% vs. 87.6%), intermediate algebra (88.7% vs. 82.9%), and counting and probability (92.8% vs. 86.1%). The methods performed similarly in geometry and pre-calculus (72.4% vs. 70.2% in geometry).

These experimental results show that while ToT and MDToC perform similarly on geometry-related and visual-understanding problems, notable differences emerge when more precise calculation steps are required. Geometry questions often hinge on spatial reasoning and visual understanding, domains in which both prompting methods perform equally well. In these tasks, the abstract thought evaluations encompassed by ToT appear sufficient to address the reasoning needed for shapes, angles, and other geometric relationships, while MDToC's

exclusive focus on calculations does not confer a distinct advantage. However, for algebra problems demanding intensive numeric manipulation, MDToC strongly outperforms ToT. This result aligns with MDToC's design, which specifically targets explicit calculation steps to enhance accuracy in computation-heavy contexts.

A.4 Problem types on CHAMP dataset

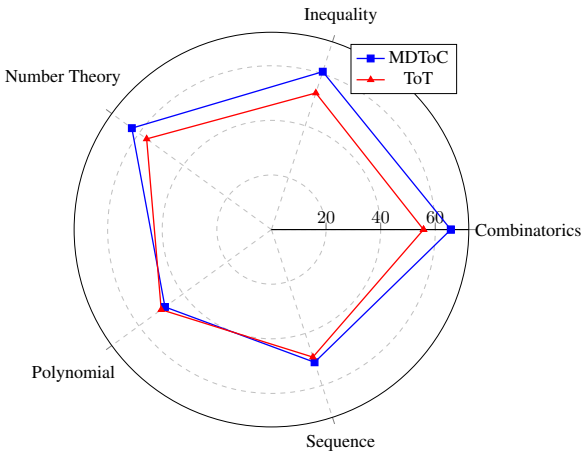

Figure 8: **Radar chart comparing our MDToC and ToT performance on the CHAMP dataset** in terms of the percentage accuracy—both evaluated with GPT-4-Turbo—across Combinatorics, Inequality, Number Theory, Polynomial, and Sequence.

A comparative analysis of MDToC and ToT across five CHAMP dataset math topics in Figure 8 shows MDToC's clear advantages in most categories. MDToC demonstrated significantly higher accuracy in Combinatorics (65.7% vs. 55.7%), Inequality (60.8% vs. 52.6%), and Number Theory (63.2% vs. 56.5%), highlighting its effectiveness in problems requiring detailed numeric calculations and methodical computation. While ToT showed a slight edge in Polynomial problems (49.9% vs. 48.2%), likely due to its strength in abstract symbolic manipulation, MDToC maintained superiority in Sequence problems (51.1% vs. 49.1%), suggesting that its explicit calculation framework better handles iterative, arithmetic-driven tasks. These results underscore how MDToC's focus on detailed numeric processes generally yields stronger performance in calculation-oriented mathematical problem-solving.