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

Abstract

Large Language Models (LLMs) show promise in mathematical problem-solving but often falter in intermediate reasoning steps, particularly in smaller models. Recent prompting techniques—including Chain-of-Thought (CoT) and Tree-of-Thought (ToT)—help refine this reasoning but frequently rely on model-specific heuristics and hand-coded knowledge. We propose MDToC (Metacognitive Dynamic Tree of Concepts), a novel framework that builds a concept tree with accuracy-verified calculations via generalized prompts, enabling robust performance across diverse mathematical domains. Our experiments on CHAMP, MATH, and Game-of-24 datasets with OpenAI LLMs (GPT-3.5, GPT-4o-mini, and GPT-4 Turbo) show that MDToC substantially outperforms standard and advanced prompting methods. On CHAMP, for example, MDToC achieves 39.5% with GPT-3.5, 48.3% with GPT-4o-mini, and 58.1% with GPT-4 Turbo—exceeding zero-shot, few-shot, CoT, ToT and annotated-hint approaches. In addition, against the advanced ToT, MDToC yields improvements of up to 7.6% on MATH and 19% on Game-of-24, illustrating its effectiveness in enhancing mathematical reasoning without requiring any hand-engineered knowledge.

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 innovations in prompting techniques have addressed limitations of earlier methods by introducing hierarchical structures for thought representation, such as Tree-of-Thoughts (ToT) [Yao *et al.*, 2024] [Long, 2023] and Graph-of-Thoughts (GoT) [Besta *et al.*, 2024] [Yao *et al.*, 2023], while incorporating evaluative components at intermediate steps. ToT organizes reasoning in a tree-like structure, generating and evaluating thought pools at each depth, while GoT represents thoughts in a graph-like structure, facilitating interconnection and transformation through various operations. These approaches have shown notable improvements in complex mathematical tasks, with ToT achieving up to 74% accuracy on Game-of-24 [Yao *et al.*, 2024] and GoT attaining approximately 89% accuracy on Sequence-Sorting-64-elements [Besta *et al.*, 2024].

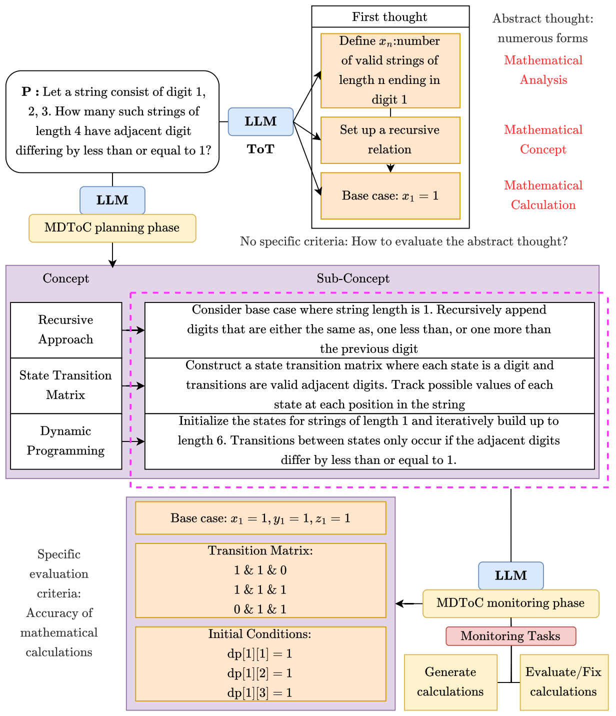
However, these methods face limitations in their evaluative mechanisms, lacking well-defined criteria as thoughts can appear in various forms such as mathematical analysis, concepts, or calculations, as shown in Figure 1. Consequently, the evaluation process often relies heavily on strong LLMs, e.g., GPT-4, resulting in approximated and unreliable evaluation scores for abstract thoughts. The challenge of standardizing diverse reasoning texts leads to potential vulnerabilities in thought selection and connection pruning, while the need for domain-specific customization underscores a lack of generalizability across problem domains.

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 a novel prompting technique named MDToC (Metacognitive Dynamic Tree of Concepts), which is structured within a three-step metacognitive framework: planning, monitoring, and reviewing. More specifically, our MDToC transforms thoughts into concepts and calculations for abstract thought evaluation. In the planning phase, MDToC utilizes a depth-two concept tree, exploring diverse mathematical and programming concepts at the first depth and contextualizing these with problem-specific information as sub-concepts at the second. This structured approach constrains the solution space, mitigating hallucination and enhancing problem-solving versatility.

The monitoring phase expands the sub-concepts in the planning phase with intermediate calculation steps, employing three LLMs for calculation generation, evaluation, and error rectification, while a fourth LLM assesses progress toward the problem's objective. In the reviewing phase, two LLMs validate potential answers against the problem's objective, eliminating invalid solutions, followed by majority voting to determine the final solution. This comprehensive framework allows for precise evaluation criteria, focused on computational accuracy and problem-specific objectives.

Our proposed MDToC prompting has shown significant efficacy in mathematical problem-solving. Particularly, using GPT-3.5, we achieved 39.5% accuracy on the CHAMP dataset [Mao *et al.*, 2024], surpassing GPT-3.5 with annotated concepts and hints by 5.1%. On the Game-of-24 task [Yao *et al.*, 2024], GPT-4o-mini with our MDToC attained 75% accuracy, outperforming GPT-4o-mini with ToT prompting by 19%. Such improvements come from the fact that our MDToC is capable of transforming abstract thoughts into concepts and calculations, enhancing reasoning text evaluation and generation. In addition, it is powered with generalized prompt sets across diverse mathematical datasets while maintaining a hierarchical prompting structure.



P : Let a string consist of digit 1, 2, 3. How many such strings of length 4 have adjacent digit differing by less than or equal to 1?
LLM
ToT
First thought
Define x_n:number of valid strings of length n ending in digit 1
Set up a recursive relation
Base case: $x_1 = 1$
Abstract thought: numerous forms
Mathematical Analysis
Mathematical Concept
Mathematical Calculation
No specific criteria: How to evaluate the abstract thought?
LLM
MDToC planning phase
Concept
Sub-Concept
Recursive Approach
State Transition Matrix
Dynamic Programming
Consider base case where string length is 1. Recursively append digits that are either the same as, one less than, or one more than the previous digit
Construct a state transition matrix where each state is a digit and transitions are valid adjacent digits. Track possible values of each state at each position in the string
Initialize the states for strings of length 1 and iteratively build up to length 6. Transitions between states only occur if the adjacent digits differ by less than or equal to 1.
Base case: $x_1 = 1, y_1 = 1, z_1 = 1$
Transition Matrix:
1 & 1 & 0
1 & 1 & 1
0 & 1 & 1
Initial Conditions:
dp[1][1] = 1
dp[1][2] = 1
dp[1][3] = 1
Specific evaluation criteria: Accuracy of mathematical calculations
LLM
MDToC monitoring phase
Monitoring Tasks
Generate calculations
Evaluate/Fix calculations

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.

The next section explores related work, which is followed by a detailed description of MDToC, comparative results on a number of standard datasets, and a concluding discussion. Hereafter, we denote annotated concepts and hints as **CH**.

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

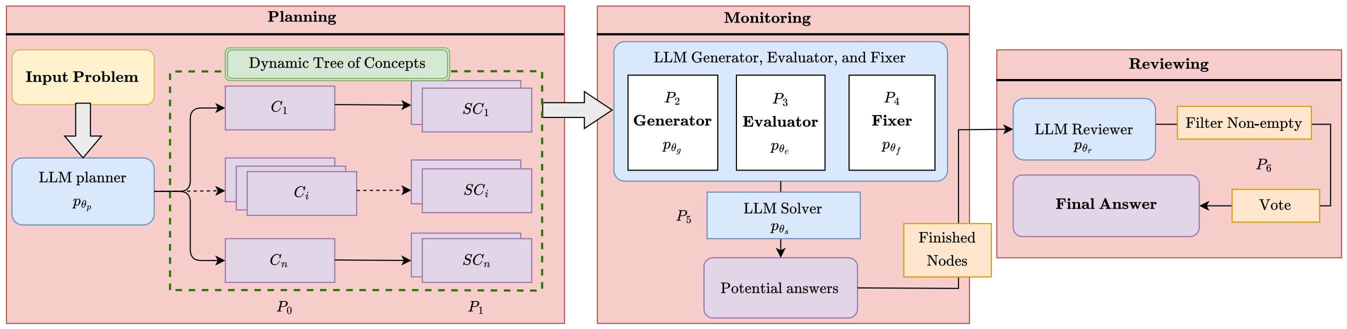


Figure 2: Proposed MDToC prompting structure. C represents the first-depth concept, while SC represents the second-depth sub-concept. P_0 and P_1 are prompts used in the planning phase shown in Figure 3, while P_2, P_3, P_4, and P_5 are prompts used in the monitoring phase given in Figure 4. Prompts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

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 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^*.

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.

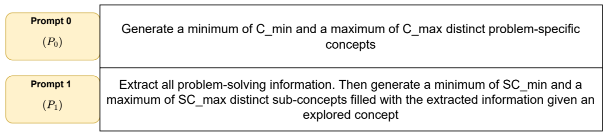
3.1 Planning

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:

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

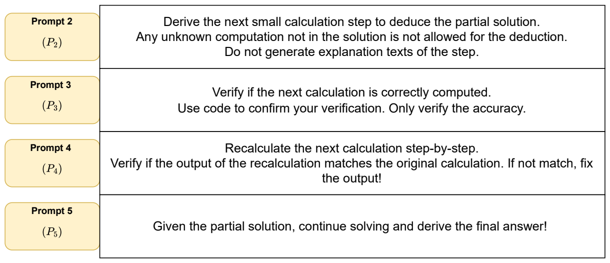
.



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.

The k-th calculation $\chi_{ijk}^{d\geq3}$ is sampled from the generator as $\chi_{ijk}^{d\geq3} \sim p_{\theta_g}(\chi_{ijk}^{d\geq3}|c_i^{d=1}, c_{ij}^{d=2}, \chi_{ijk}^{d-1\geq3})$ where $\chi_{ijk}^{d-1\geq3}$ represents previous calculations. The evaluation result $V(p_{\theta_e}, \chi_{ijk}^{d\geq3})$ for the k-th calculation is expressed as $V(p_{\theta_e}, \chi_{ijk}^{d\geq3}) \sim p_{\theta_e}(v_e, r_e|(c_i^{d=1}, c_{ij}^{d=2}, \chi_{ijk}^{d-1\geq3}, \chi_{ijk}^{d\geq3})]$, 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 $\chi_{ijk}^{d\geq3} \sim p_{\theta_g}(\chi_{ijk}^{d\geq3}|c_i^{d=1}, c_{ij}^{d=2}, \chi_{ijk}^{d-1\geq3}, \chi_{ijk}^{d\geq3}, r_e)$.

We further introduce an LLM fixer p_{θ_f} prompted with P_5 to fix the errors in the k-th calculation as $\chi_{ijk}^{d\geq3} \sim p_{\theta_f}(\chi_{ijk}^{d\geq3}|\chi_{ijk}^{d\geq3})$. 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} 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 **four additional parameters**: e, c_s, i_s, t. In particular, the parameter e determines the number of evaluation and regeneration attempts before invoking the LLM fixer. c_s and i_s specify the number of initial and 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.

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 e 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.

3.3 Reviewing

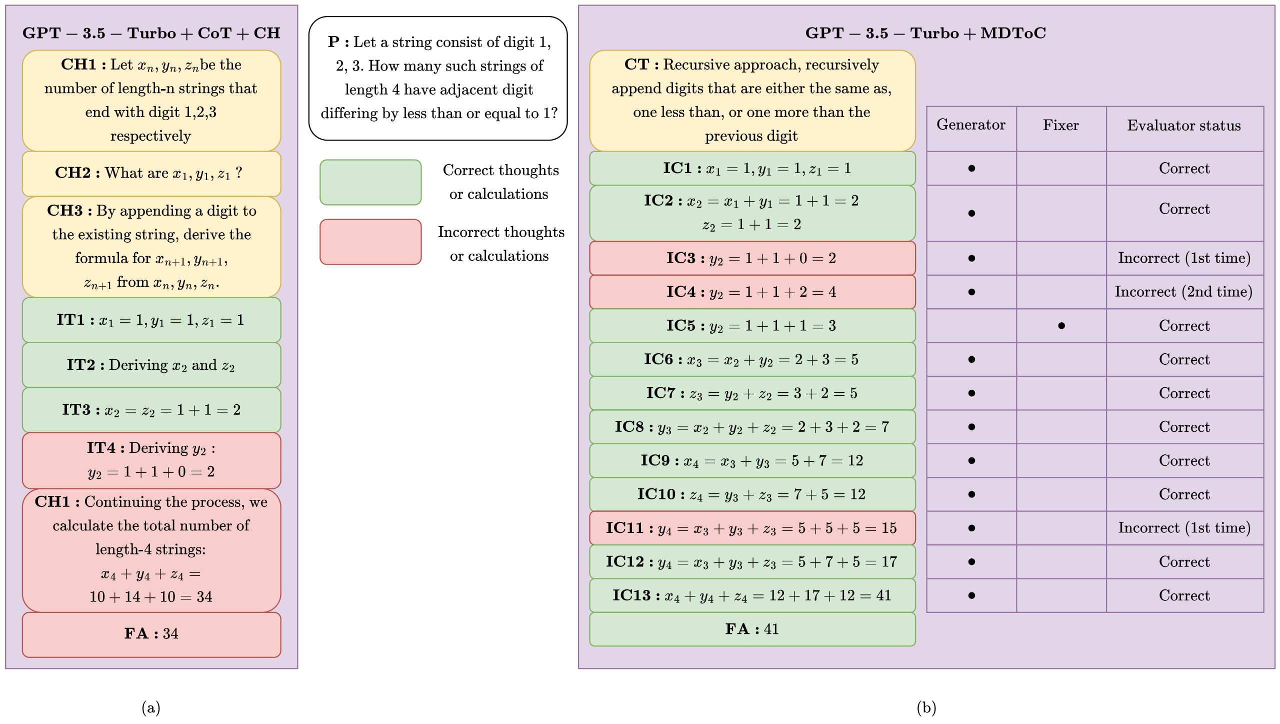
In the last phase, we obtain the results from the monitored tree and select the top-voted answers. Given a list of solutions marked by 'Finished Node' as A, we use an LLM reviewer p_{θ_r} prompted wit h P_6 shown in Figure 6 to conduct a majority voting of A. This stage involves finding the most common 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

We use three datasets to evaluate our MDToC approach. The first is CHAMP (Competition-level Dataset for Fine-Grained Analyses of LLMs' Mathematical Reasoning Capabilities) [Mao *et al.*, 2024], a collection of 270 high school-level competition math problems. This dataset encompasses five categories: number theory (80 problems), polynomial (50), sequence (50), inequality (50), and combinatorics (40), each annotated with relevant mathematical concepts and problem-specific hints. Given that MDToC is fundamentally driven by a concept tree, this dataset provides valuable insights into the interplay between contextual information and LLM reasoning in complex mathematical scenarios.

We also test MDToC on the common mathematical benchmark MATH [Hendrycks *et al.*, 2021]. This dataset compiles $12,500$ competition-level problems from sources such as AIME [AoPS, a] and AMC 10/12 [AoPS, b], encompassing a broad range of advanced topics in algebra, geometry, number theory, and more. On the other hand, to enable direct comparison with the ToT, we use the Game-of-24 dataset, a


(a)
(b)
Correct thoughts or calculations
Incorrect thoughts or calculations

P : Let a string consist of digit 1, 2, 3. How many such strings of length 4 have adjacent digit differing by less than or equal to 1?

$\mathbf{GPT-3.5-Turbo+CoT+CH}$

CH1 : Let x_n, y_n, z_nbe the number of length-n strings that end with digit 1,2,3 respectively

CH2 : What are x_1, y_1, z_1 ?

CH3 : By appending a digit to the existing string, derive the formula for x_{n+1}, y_{n+1}, z_{n+1} from x_n, y_n, z_n.

IT1 : $x_1 = 1, y_1 = 1, z_1 = 1$

IT2 : Deriving x_2 and z_2

IT3 : $x_2 = z_2 = 1 + 1 = 2$

IT4 : Deriving y_2 : $y_2 = 1 + 1 + 0 = 2$

CH1 : Continuing the process, we calculate the total number of length-4 strings: $x_4 + y_4 + z_4 = 10 + 14 + 10 = 34$

FA : 34

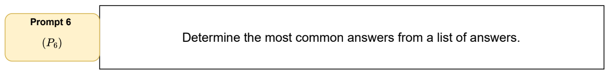
$\mathbf{GPT-3.5-Turbo+MDToC}$

CT : Recursive approach, recursively append digits that are either the same as, one less than, or one more than the previous digit

	Generator	Fixer	Evaluator status
IC1 : $x_1 = 1, y_1 = 1, z_1 = 1$	●		Correct
IC2 : $x_2 = x_1 + y_1 = 1 + 1 = 2$ $z_2 = 1 + 1 = 2$	●		Correct
IC3 : $y_2 = 1 + 1 + 0 = 2$	●		Incorrect (1st time)
IC4 : $y_2 = 1 + 1 + 2 = 4$	●		Incorrect (2nd time)
IC5 : $y_2 = 1 + 1 + 1 = 3$		●	Correct
IC6 : $x_3 = x_2 + y_2 = 2 + 3 = 5$	●		Correct
IC7 : $z_3 = y_2 + z_2 = 3 + 2 = 5$	●		Correct
IC8 : $y_3 = x_2 + y_2 + z_2 = 2 + 3 + 2 = 7$	●		Correct
IC9 : $x_4 = x_3 + y_3 = 5 + 7 = 12$	●		Correct
IC10 : $z_4 = y_3 + z_3 = 7 + 5 = 12$	●		Correct
IC11 : $y_4 = x_3 + y_3 + z_3 = 5 + 5 + 5 = 15$	●		Incorrect (1st time)
IC12 : $y_4 = x_3 + y_3 + z_3 = 5 + 7 + 5 = 17$	●		Correct
IC13 : $x_4 + y_4 + z_4 = 12 + 17 + 12 = 41$	●		Correct

FA : 41

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 $e = 2$ attempts, a threshold chosen to reduce the risk of model "hallucinations" (erroneous or fabricated steps).



Determine the most common answers from a list of answers.

Figure 6: Prompts for the reviewing phase.

mathematical puzzle where players use 4 given integers and basic arithmetic operations to reach 24. We use a subset of the 100 relatively difficult problems, indexed from 901 to 1000, mirroring the evaluation dataset in the ToT setting [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 $SC_{max} = 2$. 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 $e = 2$, $c_s = 2$, $i_s = 1$, and $t = 10$, emphasizing in-depth concept decomposition and analysis. Game-of-24 employs $e = 2$, $c_s = 10$, $i_s = 2$, and $t = 4$, focusing on broader analysis of calculation combinations.

4.3 Model Configuration

In our experiments, we exclusively use OpenAI LLMs. During the planning phase, we leverage GPT-4o to maximize concept diversification. In the review phase, we again utilize GPT-4o to standardize different responses and obtain accurate voting. For the monitoring phase, we deploy GPT-4o-mini specifically for the fixer component p_{θ_f} (see Fig. 2) to maintain the fixing accuracy while optimizing cost efficiency. Meanwhile, for all other components including the generator, evaluator, and solver (see Fig. 2), we use three GPT models—GPT-3.5-Turbo, GPT-4o-mini, and GPT-4-Turbo—so that we can compare our approach directly with other prompting methods. As such, GPT-3.5-Turbo, GPT-4o-mini and GPT-4-Turbo are used for the generator, evaluator, and solver in the monitoring phase of the GPT-3.5-Turbo+MDToC, GPT-4o-mini+MDToC, and GPT-4-Turbo+MDToC schemes, respectively, while all of these three schemes use GPT-4o for the planning and reviewing phases, and GPT-4o-mini is used for the fixer component.

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 uses less than 1% of total tokens for both CHAMP and MATH datasets and approximately 2% for Game-24. In the fixer component, GPT-4o-mini consumes 23,428, 21,599, and 1,945 tokens for CHAMP, MATH, and Game-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 three 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 stands for Game-of-24)

GPT	Component	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-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, and GPT-4-Turbo 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.

Prompting	GPT Model		
	3.5-Turbo	4o-mini	4-Turbo
0-shot	28.5	36.2	41.9
CoT	29.6	36.5	37.8
5-shot	34.8	38.7	43.1
CoT + CH	34.4	42.3	53.0
1/3 sln	33.7	43.4	53.7
ToT	31.7	37.8	52.7
Our MDToC	**39.5**	**48.3**	**58.1**

Our MDToC demonstrates superior performance across all models, achieving 39.5% (GPT-3.5-Turbo), 48.3% (GPT-4o-mini), and 58.1% (GPT-4-Turbo). These results represent substantial improvements over ToT, 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, and 68.6% for GPT-4-Turbo 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). The ToT method surpasses standard few-shot prompts for the two more advanced models, pushing GPT-4-Turbo to 80.4% — a notable jump from 70.3% under CoT.

Even so, MDToC outperforms all these strategies, achieving 60.8% for GPT-3.5-Turbo, 83.8% for GPT-4o-mini, and 86.6% for GPT-4-Turbo. Compared to ToT, 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 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

Prompting	GPT Model		
	3.5-Turbo	4o-mini	4-Turbo
0-shot	45.7	71.4	68.6
CoT	48.6	72.4	70.3
5-shot	54.3	77.1	79.3
ToT	53.2	78.5	80.4
Our MDToC	**60.8**	**83.8**	**86.6**

4.6 Game-of-24 evaluation

Table 4 compares five prompting methods — 0-shot, CoT, 5-shot, ToT, and MDToC — across three LLMs (GPT-3.5-Turbo, GPT-4o-mini, and GPT-4-Turbo) on the Game-of-24 dataset. Simple methods like 0-shot and CoT yield very low scores (between 2–4% accuracy), while providing multiple examples (5-shot) improves performance moderately (6–10%). ToT then brings a significant jump, especially for GPT-4o-mini (56%) and GPT-4-Turbo (74%). However, MDToC achieves the highest accuracies for all three LLMs, peaking at 85% for GPT-4-Turbo and outperforming ToT by substantial margins (e.g., 11% higher for GPT-4-Turbo). This underscores the critical role of the evaluator p_{θ_e} and the fixer p_{θ_f} in our methodology. This component corrects erroneous intermediate expressions involving 4 numbers of Game-of-24, which is crucial for generating subsequent calculations accurately and mitigating hallucinations.

5 Discussion

5.1 Problem types on MATH 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

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

Prompting	**GPT Model**		
	3.5-Turbo	4o-mini	4-Turbo
0-shot	2	3	4
CoT	3	3	4
5-shot	6	8	10
ToT	19	56	74
Our MDToC	**30**	**75**	**85**

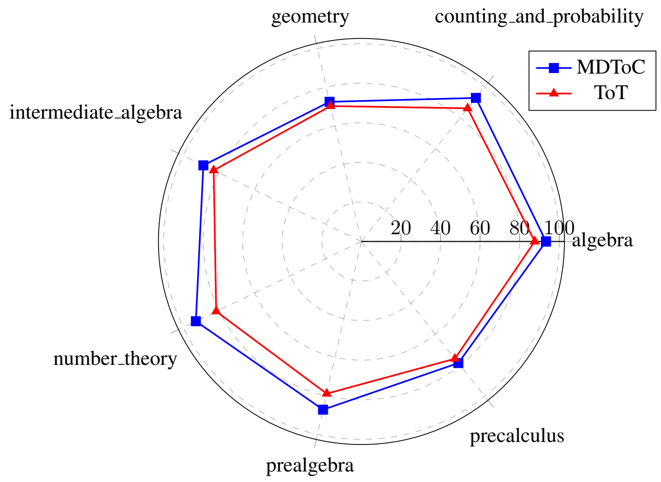


Figure 7: **Radar chart comparing our MDToC and ToT performance on the MATH dataset** in terms of the percentage accuracy—both evaluated with GPT-4-Turbo—on various math problems of this dataset.

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.

5.2 Problem types on CHAMP dataset

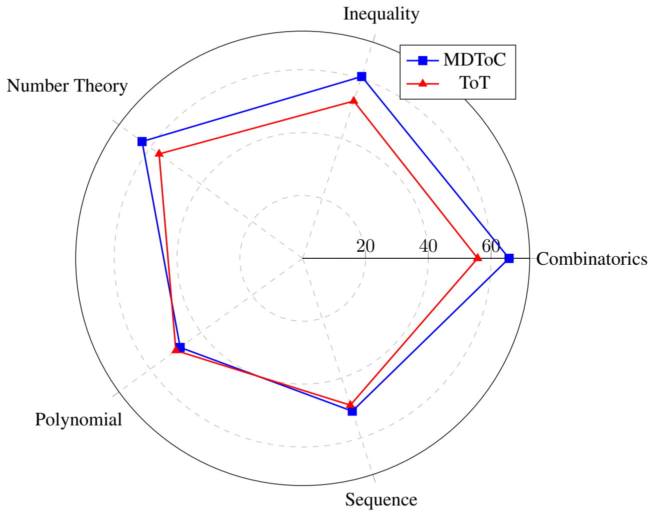


Figure 8: **Radar chart comparing our MDToC and ToT performance on the CHAMP dataset** in terms of the percentage accuracy—both evaluated with GPT-4-Turbo—across Combinatorics, Inequality, Number Theory, Polynomial, and Sequence.

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.

6 Conclusion

Our proposed MDToC framework significantly enhances the mathematical reasoning capabilities of LLMs across diverse domains through its structured metacognitive approach of planning, monitoring, and reviewing. The framework outperforms existing techniques like CoT and ToT, achieving up to 11% higher accuracy than ToT on Game-24 and showing consistent improvements across CHAMP, MATH, and Game-24 datasets. While MDToC excels in calculation-intensive tasks through its dynamic concept structuring, iterative error correction, and majority voting, it faces challenges with sequence and geometry problems that require pattern recognition and visual understanding. Despite these limitations, MDToC emerges as a powerful solution for advancing mathematical reasoning in LLMs, establishing a foundation for future research in complex problem-solving.

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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