# Understanding When Tree of Thoughts Succeeds: Larger Models Excel in Generation, Not Discrimination

**Anonymous ACL submission**

## Abstract

Tree of Thoughts (ToT) is a reasoning strategy for Large Language Models (LLMs) that employs a generator to suggest reasoning steps and a discriminator to decide which steps to implement. ToT demonstrates strong performance on reasoning tasks, often surpassing simple methods such as Input-Output (IO) prompting and Chain-of-Thought (CoT) reasoning. However, ToT does not consistently outperform such simpler methods across all models, leaving large knowledge gaps on the conditions under which ToT is most beneficial. In this paper, we analyze the roles of the generator and discriminator separately to better understand the conditions when ToT is beneficial. We find that the generator plays a more critical role than the discriminator in driving ToT's success. While using even a smaller model as the discriminator, scaling the generator leads to notable improvements in ToT performance, whereas scaling the discriminator with a fixed generator yields only marginal gains. Our results show that models across different scales exhibit comparable discrimination capabilities, yet differ significantly in their generative performance in the setup of ToT.

## 1 Introduction

Since the introduction of CoT (Wei et al., 2022), which has enhanced the reasoning capabilities of LLMs, numerous new prompting-based methods have been proposed to further support LLM-based reasoning (Besta et al., 2024a; Sel et al.; Radha et al., 2024; Chen et al., 2023; Ranaldi et al., 2024; Cao et al., 2023; Mo and Xin, 2024; Wang et al., 2023a; Zhou et al.; Khot et al.). Among these, the ToT method proposed by Yao et al. (2023) extends the CoT approach into a tree search framework, demonstrating its potential to enhance the reasoning performance of state-of-the-art LLMs, such as GPT-4 (Achiam et al., 2023), across complex reasoning tasks, including the Game of 24, Creative Writing, and Mini Crosswords (Yao et al., 2023).

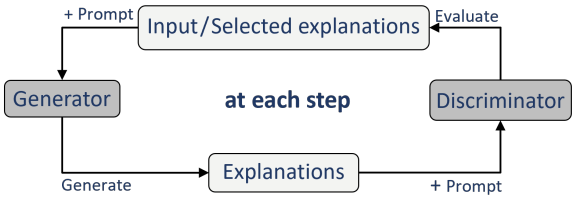



Figure 1: Core Mechanism of ToT. ToT employs a generator to suggest reasoning steps and a discriminator to decide which steps to implement.

ToT is a prompting framework that encourages the model to generate and self-evaluate intermediate reasoning steps. Theoretically, ToT offers advantages over simpler methods like Input-Output (IO) prompting and CoT reasoning through extensive exploration (via the generator) and optimal selection mechanisms (via the discriminator), as shown in Figure 1. However, when ToT is applied to a wider range of LLMs and task types, it has been found to adversely affect the inherent reasoning abilities of weaker LLMs (Duan et al., 2024). This reveals that the practical implications of ToT across different model scales remain underexplored. Additionally, Chen et al. (2024) highlight that advanced planning methods like tree search require high-quality discriminators (accuracy $\geq 90\%$) to outperform simple re-ranking methods. However, the current discriminative capabilities of most LLMs fall short of this threshold, limiting the effectiveness of such advanced techniques. Their work highlights the importance of the discriminator's capability when using tree-search-based methods; however, their conclusions are drawn based on the evaluation of only one single generator (i.e., CodeLlama-13B-Inst (Roziere et al., 2023)).

This study aims to gain deeper insights into whether specific scales of LLMs can benefit from ToT when addressing problems of mathematical and logical reasoning. We systematically compare

the performance of ToT against baseline methods, including IO prompting and CoT reasoning, to evaluate ToT's performance across various model scales on challenging mathematical reasoning tasks and natural language-based logical reasoning problems, aiming to identify the conditions under which ToT offers significant improvements. Our investigation is framed around three research questions: (1) Does scaling the size of the **generator** improve the performance of ToT? (2) Does scaling the size of the **discriminator** improve the performance of ToT? (3) Under which conditions does ToT outperform IO and CoT?

Our key findings are:

- ToT tends to benefit larger models more than smaller ones.
- Scaling the size of the generator results in significant improvements while scaling the size of the discriminator provides only marginal gains.
- Further experiments show that while models of different scales exhibit similar discriminative abilities when applying ToT, their generation performance varies significantly.

By uncovering effective conditions for the successful application of ToT, we aim to aid users in making informed decisions about when to employ the ToT framework and when to restrain to simpler methods to avoid unnecessary resource costs.

## 2 Background: Tree of Thoughts

ToT, as illustrated in Figure 1 and proposed by Yao et al. (2023), is a search-based approach designed to enhance reasoning in LLMs by extending the stepwise reasoning of the CoT method (Wei et al., 2022) into a tree search paradigm. The core mechanism of ToT involves two actors: a generator that proposes multiple intermediate reasoning steps, which are then evaluated by a discriminator to select the most promising ones for subsequent steps in the search process.

ToT systematically assesses candidate explanations using search algorithms such as Breadth-First Search (Moore, 1959) and Depth-First Search (Tarjan, 1972) to find the optimal path to the best solution. In this context, several terms are crucial for understanding the ToT framework. **Generation** refers to the process by which a **generator** produces intermediate steps or explanations, also known as exploration or expansion of the search tree. Yao et al. (2023) introduces two key methods for evaluating these explanations. The **Value** method quantifies each explanation independently, converting it into scalar values (e.g., 1-10) or categorizations (e.g., confident/likely/impossible), based on various evaluative criteria like forward-looking simulations or common sense. The **Vote** method compares all generated explanations through a voting process, selecting the most promising partial solutions when direct assessment of reliability is difficult.

Explanations generated by the LLM can be categorized as valid or invalid. **Valid explanation** adhere to game rules, while **invalid explanation** violate them. Among valid explanations, **viable explanations** (or promising explanations) are those that potentially lead to a solution. **Inviable explanations** (or unpromising explanations) are valid but unlikely to lead to success. These relationships are depicted in Figure 2.

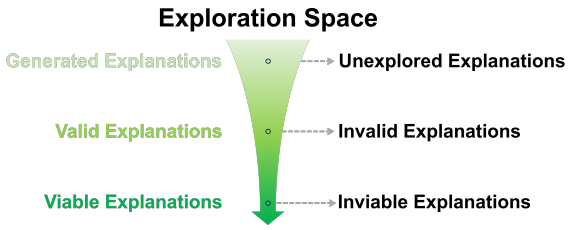



Figure 2: Diagram of relationships among different types of explanations in ToT.

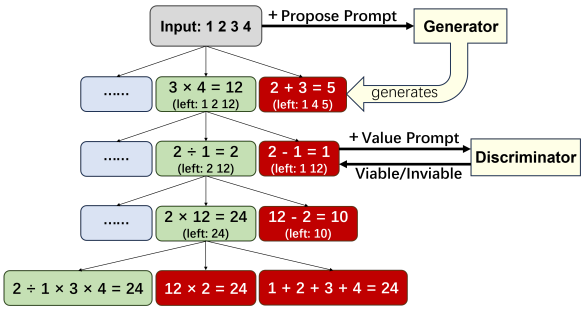
The **search space** or exploration space refers to the number of choices available at each step. The LLM responsible for evaluating candidate explanations is referred to as the **discriminator** or evaluator. Together, these components form the foundation of ToT's reasoning and search mechanisms.

## 3 Experimental Setup

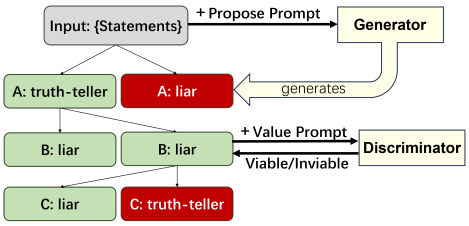
This section outlines the specific experimental setup, including the tasks and metrics used for evaluation, selected LLMs, and the details of ToT implementation for each task. It also covers the baselines used for comparison and the specific prompts employed in the experiments.

### 3.1 Tasks

This study selects two challenging reasoning tasks — Game of 24 and Knights and Knaves — for investigating the efficacy of the ToT approach in mathematical and natural language reasoning tasks.




(a) Game of 24




(b) Knights and Knaves

Figure 3: Illustration of Game of 24 and Knights and Knaves under ToT setting. The generator proposes possible intermediate steps, which will be evaluated by the discriminator. We denote the selected/discarded intermediate partial solutions in green/red.

Game of 24 is a mathematical logic puzzle characterized by high decision complexity (refer to Section 3.1.1 for further details). According to the results reported by Yao et al. (2023), even state-of-the-art LLMs such as GPT-4 (Achiam et al., 2023) using baseline methods achieve a success rate of less than 10%. In contrast, Knights and Knaves is a natural language-based logic reasoning task that involves extensive hypothesis generation, inference, and backtracking. Despite having a decision complexity of only $2^{\#\text{characters}}$, Mondorf and Plank (2024)'s findings indicate that LLMs employing baseline methods achieved an accuracy of no more than 30%. Therefore, there is substantial room for improvement in the reasoning capabilities of models for both tasks.

### 3.1.1 Game of 24

**Rule** The Game of 24 is a mathematical reasoning challenge that presents an arithmetic task. The objective is to manipulate exactly four integers within the range of 1 to 13, using basic arithmetic operations, (i.e., $+, -, \times, \div$), to achieve a final result of 24. For instance, given the numbers 1, 2, 3, and 4, a possible solution could be formulated as $(1 + 3) \times (2 + 4) = 24$. As Illustrated in Figure 3 (a), in each round of the game, the LLM receives an input of four integers and is expected to provide the correct solution corresponding to this input. This task may possess multiple valid solutions.

Given four numbers, they can be permuted in various sequences. For each permutation, there are three interstitial positions where an operator can be inserted, with four possible operators for each position. Multiplying all the combination possibilities together, we can get: Decision Complexity = $4! \times 4^3 = 1,536$.

**Dataset** In order to facilitate a more effective comparison between our experimental results and those of Yao et al. (2023), we utilize the same dataset proposed in their work. This dataset comprises 1,362 games, which are arranged in ascending order of difficulty based on the time taken by humans to solve them. However, this dataset does not include the solutions to the problems. Therefore, we generate all feasible solutions for each task using an algorithm for our experiments (see Appendix C).[1]

### 3.1.2 Knights and Knaves

**Rule** Knights and Knaves puzzles are a class of logical puzzles in which each character is either a "Knight" or a "Knave". The fundamental rule of these puzzles is that a Knight always tells the truth, meaning that any statement made by a Knight is logically consistent with the facts. In contrast, a Knave always lies, meaning that every statement made by a Knave is false. The objective of a LLM agent is to logically deduce the identity of each character based on their statements (Figure 3 (b)).

**Dataset** We utilize the dataset published by Mondorf and Plank (2024), which comprises a total of 2,400 distinct tasks. These tasks are divided into four subsets based on varying numbers of characters (indicating difficulty levels), with $n = 3, 4, 5, 6$. Each subset contains 600 problems, and every task in the dataset has a unique solution. We use the subset with 3 characters in our experiments.

## 3.2 Evaluation Metrics

Following the work of Yao et al. (2023), we use *average success rate* as metric to assess the results of the Game of 24 and Knights and Knaves.

**Average Success Rate** According to Yao et al. (2023)'s work, the average success rate can be de-

[1]The algorithm is also included in the anonymous repo.

fined as follows:

$$\text{Average Success Rate} = \frac{\sum_{\text{tasks}} \frac{\text{\# correct answers}}{\text{\# outputs}}}{\text{\# tasks}} \quad (1)$$

The average success rate is in the range [0, 1], with 1 indicating that all tasks are successful and 0 indicating that all tasks fail.

Note that the average success rate eliminates the advantage of multiple attempts for the same problem in IO, CoT, and ToT methods, allowing for direct comparison of results across different approaches. This also facilitates comparison with accuracy metrics reported in other studies, even when the number of attempts per task differs.

### 3.3 Language Models

Based on the classification of language model scales in Yang et al. (2024), six open-source language models at three different scales are used in the experiments of this study, which are:

- smaller models (with <10B parameters): gemma-2b-it (Team et al., 2024), Llama-3-8B-Inst (AI@Meta, 2024) and Llama-3.1-8B-Inst (Dubey et al., 2024),
- medium models (with $\geq$ 10B and <50B parameters): gemma-2-27b-it (Team, 2024),
- larger models (with $\geq$ 50B parameters): Llama-3-70B-Inst (AI@Meta, 2024) and Llama-3.1-70B-Inst (Dubey et al., 2024).

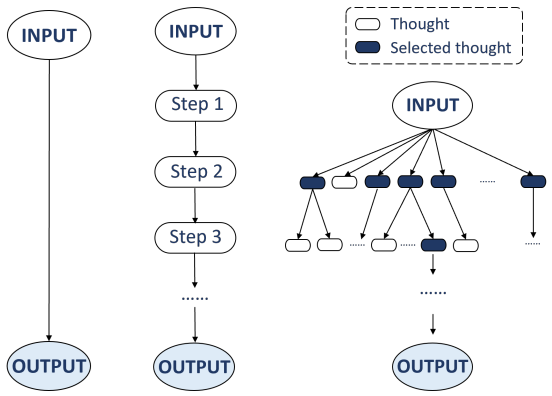
In the experiments for **RQ1** and **RQ2**, we use Llama-3.1-8B-Inst, gemma-2-27b-it and Llama-3.1-70B-Inst. In the experiments for **RQ3**, all models except gemma-2-27b-it are included. All models used are instruction-tuned models. We query the API provided by Hugging Face[2] for model inference. We employ *temperature sampling* as the models' generation strategy in all runs. Unless otherwise specified, the decoding temperature for the model in all experiments is set to 0.7.

### 3.4 Baselines & ToT Oracle

#### 3.4.1 Baselines

**IO Prompting** Input-output prompting, or immediate output (also called direct prompting, as shown in Figure 4 (a)), represents the most direct, fundamental, and commonly used prompting method for guiding LLMs to address problems. In basic IO prompting, the LLM immediately provides a final response upon receiving the initial user prompt, without outputting any intermediate reasoning steps.

[2]https://huggingface.co/models




(a) IO (b) CoT (c) ToT

Figure 4: Illustration of IO, CoT and ToT.

**CoT** The CoT method, as illustrated in Figure 4 (b) and introduced by Wei et al. (2022), enhances the IO prompting approach by incorporating explicit intermediate reasoning steps beyond the input and output. This method improves the problem-solving capabilities of LLMs, enabling them to address complex problems incrementally.

#### 3.4.2 ToT Oracle

In this study, we decompose the ToT framework into two distinct modules — the generator and the discriminator — for separate examination. In order to independently study how each module affects the overall performance of ToT, we utilise an oracle generator and oracle discriminator, respectively, with controllable accuracy to replace the LLM agent of one of the modules.

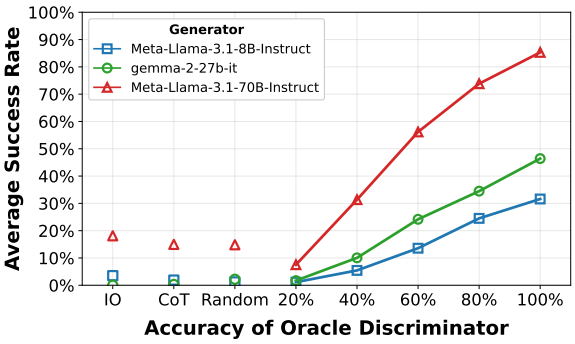
The oracle generator, like the LLM agent, produces $\#generation$ candidate explanations, with each explanation being viable with probability $p$ (inviable with probability $(1-p)$). Analogously, the oracle discriminator selects $\#selection$ explanations from the candidate pool using the same approach. If no viable explanation can be generated or selected due to earlier errors, or if the oracle discriminator fails to choose the required explanation, the task is considered a failure. This process allows us to statistically control the oracle's accuracy at a threshold of $p$. Additionally, we use a random discriminator that selects $\#selection$ explanations randomly from the candidate pool.

Due to the high task complexity of Game of 24, we consistently apply a few-shot prompt in IO, CoT and ToT to help LLMs understand this task. In contrast, for Knights and Knaves, we uniformly employ a zero-shot prompt. For detailed baselines, ToT configurations, and the prompts usage, please refer to Appendix A.1.

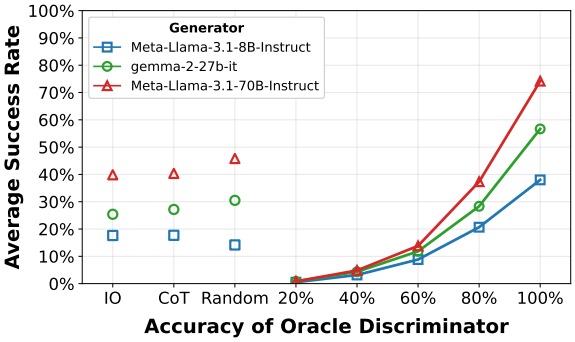
## 4 Experiment and Results

### 4.1 RQ1: Does scaling the size of the generator improve ToT's performance?

**Generation Performance Evaluation** To address RQ1, we employ three different LLMs of varying size as generators and control the accuracy of the oracle discriminator (with accuracy levels of 20%, 40%, 60%, 80%, 100%, and a random discriminator) to investigate the impact of the generator while fixing the discriminator at a certain level. At 100%, we can determine the upper bound of ToT's performance with different generators.

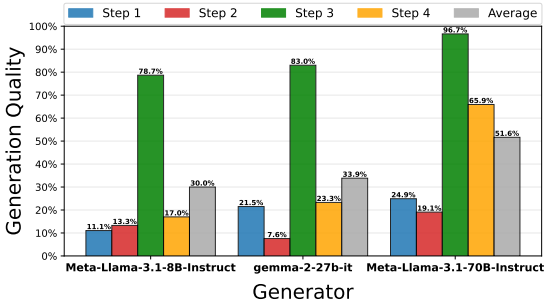



(a) Game of 24

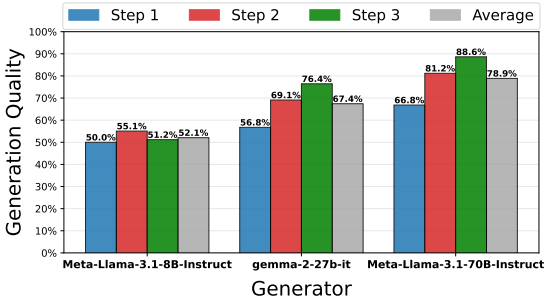



(b) Knights and Knaves

Figure 5: Impact of different models as generators on the overall performance of ToT against oracle discriminators. The lines plot illustrates the average success rate when paired with oracle discriminators. We also plot the performance of IO, CoT, and in combination with a random discriminator.




(a) Game of 24




(b) Knights and Knaves

Figure 6: The average generation quality at each step and the overall average generation quality when different models are used as generators in both tasks. The values represent the proportion of unique viable explanations among the candidate explanations at each step, where higher values indicate better generation quality. We employ the oracle discriminator with 100% accuracy to eliminate the influence of erroneous preceding steps on subsequent steps, allowing for a clearer analysis of the generator's performance.

Figure 5 displays the performance of LLM generators of varying sizes on Game of 24 and Knights and Knaves, respectively. Besides the performance of the ToT framework, we also plot the performance of the models using IO, CoT, and a random discriminator within the ToT framework as baseline references. The results show that regarding the baselines, CoT is similar to or sometimes even worse than IO on Game of 24. Regarding ToT, for all three LLMs and both datasets, as the performance of the oracle discriminator increases, the overall performance of ToT increases, and substantially outperforms the baseline methods for the Game of 24. For that game, the advantages of more powerful generators (compare Llama-3.1-70B-Inst vs Llama-3.1-8B-Inst) become increasingly apparent as the accuracy of the oracle discriminator improves. A similar trend can be observed in Knights and Knaves, although the difference is less pronounced compared to Game of 24. Moreover, on

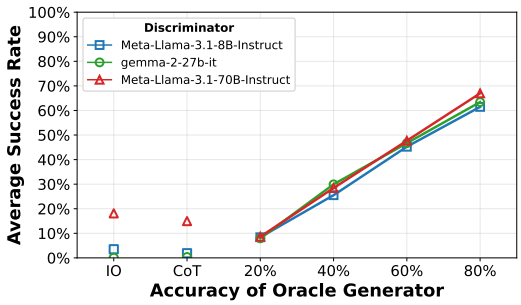
Knights and Knaves ToT is often worse than CoT, and only outperforming CoT at high discriminator accuracy levels. We hypothesize this is primarily due to the smaller decision complexity of Knights and Knaves in contrast to the larger search space in Game of 24 (see Appendix B), which makes it hard to solve in the IO and CoT settings.

**Step-wise Generation Quality** To further investigate the performance differences when using various models as generators, we measure the quality of the intermediate steps generated by each model based on the proportion of unique viable explanations among the candidate steps. As shown in Figure 6, the performance differences stem from the varying quality of candidate explanations generated by different models. The stronger the model, the higher the proportion of viable explanations it generates, statistically increasing the likelihood of selecting viable explanations and thus leading to a higher success rate.

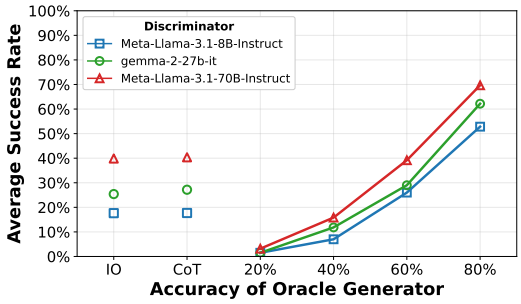
Figure 6 (a) illustrates the generation quality of three models acting as generators in the Game of 24. Upon examining the average generation quality at each step individually, we observe that the overall generation quality is low in the first two steps and the final step, remaining below 25%. However, at the third step, the quality notably increases, with even Llama-3.1-8B-Inst surpassing 78%. This is because the search space for the first two steps is quite large (48 and 24, respectively), making it difficult to find intermediate steps that result in 24. In contrast, the third step only requires selecting the correct operation sequence, which is relatively straightforward. Furthermore, it is evident that Llama-3.1-70B-Inst demonstrates significantly higher generation quality in the final step compared to smaller models. A detailed analysis of the model responses reveals that smaller models frequently repeat the few-shot examples provided in the prompt when attempting to combine three equations into one. This indicates that merging three equations into one according to the prompt, without being influenced by the prompt examples, presents a challenge for smaller models (see Appendix A.1 for prompt details).

Figure 6 (b) depicts the generation quality of the same three models in Knights and Knaves. It is evident that Llama-3.1-8B-Inst performs similarly to a random generator in Knights and Knaves, with its generation quality only slightly exceeding 50% at each step. As the model size increases, the overall generation quality also improves progressively.

### 4.2 RQ2: Does scaling the size of the discriminator improve the performance of ToT?




(a) Game of 24




(b) Knights and Knaves

Figure 7: Impact of different models as discriminators on the overall performance of ToT. The lines plot on the right side of the figures illustrate the average success rate when paired with oracle generators. The left side of the figures provide a baseline comparison of the performance of the three models using IO and CoT.

**Evaluating the Discrimination Performance of LLMs** To address RQ2, we use three different sizes of LLMs as discriminators and vary the accuracy of the oracle generator (20%, 40%, 60%, 80%) to control the influence of generators on the experiments. We restrain from using a perfect generator with 100% accuracy as this would make the discriminator redundant. Unless specified otherwise, in Game of 24, the oracle generator produces 10 candidate explanations at each step, while in Knights and Knaves, it generates two, as each character in Knights and Knaves has only two possible identities, whereas the Game of 24 has a much larger game tree.[3] Figure 7 displays the performance of different-sized LLMs as discriminators in Game of 24 and Knights and Knaves, respectively.

[3]For theoretical calculations of the branching factor in Game of 24, refer to the Appendix B.

On the left side of the figures, we provide baseline performance using IO and CoT for comparison.

In both tasks, it can be observed that as the performance of the oracle generator improves, the overall effectiveness of ToT also increases. Moreover, similar trends can be observed across the two datasets (with ToT having a lesser impact on Knights and Knaves, further discussed later). Notably, unlike the generator module, the performance differences between models serving as discriminators **do not significantly widen with the increasing accuracy of the oracle generator**; instead, they remain relatively stable and consistent. In the case of Knights and Knaves, slight differences among the three model sizes emerge when the oracle generator's accuracy exceeds 20%. However, in the Game of 24, regardless of the oracle generator's accuracy, **the three models show almost no difference when used as discriminators**. This suggests that the discriminative ability of Llama-3.1-70B-Inst is not substantially superior to that of Llama-3.1-8B-Inst. Overall, we find that the generator module plays a more significant role in ToT's performance.

### 4.3 RQ3: Under which conditions does ToT outperform IO and CoT?

From Figures 5 and 7, it is evident that ToT does not always outperform baseline methods. Therefore, this section aims to explore the conditions under which LLMs using ToT leads to superior performance compared to baseline approaches.

**Task Choice Matters** As discussed in Sections 4.1 and 4.2, Game of 24 benefits more from using ToT than Knights and Knaves. As shown in Table 1, when using a strong generator for Game of 24, ToT performance significantly outperforms the baselines (33.7% vs. 3.5%). While using a weaker generator, ToT doesn't lead to performance gain, which will be discussed later. For the baseline methods, they generally fail at such games with a high complexity level. For instance, in the Game of 24, Llama-3-8B-Inst achieve average success rates below 3% with IO and CoT setups. Llama-3-70B-Inst achieves a success rate of 3.47% under IO prompting and 10.44% under CoT reasoning, as shown in Table 1. In contrast, in the game of Knights and Knaves, IO and CoT achieve relatively high average success rates. For instance, Llama-3.1-8B-Inst achieves an average baseline success rate of around 17%, while Llama-3.1-70B-Inst reaches approximately 40%. The stronger performance of LLMs using IO and CoT in the Knights and Knaves task can possibly be attributed to its lower decision complexity. At each step, the generator only needs to select one out of the available characters (3 maximum) and there are only two possible identities (Truth-teller or Liar). Whereas in the game of 24, the model needs to select from the available numbers, 4 basic arithmetic operations and the ways to combine them. The large search space makes it difficult for IO and CoT to fully explore, leading to poor performance.

| Task | Generator | Discriminator | Avg. Success Rate |
|---|---|---|---|
| Game of 24 | 8B | — (IO) | 2.28% |
| | | — (CoT) | 1.57% |
| | | 2B | 0.18% |
| | | 8B | **1.29%** |
| | | 70B | **1.85%** |
| | 70B | — (IO) | 3.47% |
| | | — (CoT) | 10.44% |
| | | 2B | 10.76% |
| | | 8B | **33.38%** |
| | | 70B | **33.76%** |
| Knights and Knaves | 8B | — (IO) | 17.63% |
| | | — (CoT) | 17.70% |
| | | 8B | **20.50%** |
| | | 70B | **21.33%** |
| | 70B | — (IO) | 39.87% |
| | | — (CoT) | 40.37% |
| | | 8B | **52.00%** |
| | | 70B | **52.00%** |

Table 1: Overall performance of ToT when using LLMs as both the generator and discriminator. When fixing the generator, using a larger model as discriminator only gives marginal gain. ToT provides more benefits when using a larger model as a generator. [4]

**Strong Generator Matters** To align the experiments with real-world applications, we conduct experiments using LLMs as both the generator and discriminator modules in ToT and compare them with baseline methods. The results, as shown in Table 1, indicate that **when the discriminator remains constant, enhancing the generator's capability can significantly increase the average success rate**. In Game of 24, this improvement exceeds 18 times, highlighting that the generator's ability has a greater impact on ToT's performance than the discriminator's. Whereas in Knights and Knaves, enhancing the generator result in a performance increase of over 2.4 times. Additionally,

[4] In Game of 24, "2B" refers to "gemma-2b-it", "8B" refers to "Llama-3-8B-Inst", and "70B" refers to "Llama-3-70B-Inst". In Knights and Knaves, "8B" refers to "Llama-3.1-8B-Inst" and "70B" refers to "Llama-3.1-70B-Inst"

when modifying the discriminator while keeping the same generator, the performance in the Game of 24 improved by no more than 13%. In Knights and Knaves, there was no observable difference in the discriminatory abilities between the Llama-3-8B-Inst and Llama-3-70B-Inst models. These findings suggest that **the generator's capacity has a more significant impact on ToT's performance compared to the discriminator. When applying ToT, the advantage of larger models over smaller ones primarily lies in their superior generative capabilities rather than their discriminatory abilities.**

In Game of 24, when the generator is relatively weak (as in the case of Llama-3-8B-Inst), ToT performs significantly worse than IO prompting. However, when the generator is sufficiently strong (as with Llama-3-70B-Inst), ToT can outperform baseline methods, even when gemma-2b-it is used as the discriminator. This finding supports the claim made in Appendix B.2 of Yao et al. (2023) that the bottleneck in Game of 24 lies in explanation generation. In the Knights and Knaves task, regardless of whether Llama-3.1-8B-Inst or Llama-3.1-70B-Inst is used as the generator, the ToT method consistently outperforms baseline methods. We attribute this difference from the Game of 24 to the lower complexity of Knights and Knaves, which allows Llama-3.1-8B-Inst to act as a generator and still surpass the baseline methods. We believe that if the generator can provide high-quality candidate explanations, using a smaller model like Llama-3-8B-Inst as the discriminator is sufficient to significantly enhance the performance of LLMs utilizing ToT, compared to baseline methods.

## 5 Related Work

**Scaffolding to Enhance Reasoning of LLMs** Shwartz et al. (2020) showed that language models can generate chain-of-thought answers with the help of scaffolding by asking clarification questions. Since the introduction of Scratch Pading (Nye et al., 2021), CoT (Wei et al., 2022) and zero-shot CoT (Kojima et al., 2022), numerous scaffolding methods have been proposed to further support LLM reasoning. These can be broadly categorized into graph-based approaches — such as SC-CoT (Wang et al., 2023b), ToT (Yao et al., 2023), GoT (Besta et al., 2024a), and MindMap (Wen et al., 2024) — and non-graph-based approaches like AoT (Sel et al.), IoT (Radha et al., 2024), XoT (Ding et al., 2024), PoT (Chen et al., 2023) etc.

**Evaluation of Scaffolding Methods** Duan et al. (2024) introduced a novel benchmark aimed at evaluating the strategic reasoning abilities of LLMs in game-theoretic scenarios. Besta et al. (2024b) conducted an extensive analysis of various reasoning structures employed by LLMs, exploring the relative strengths and limitations of chain-based, tree-based, and graph-based reasoning strategies. In contrast, our study focuses on an in-depth analysis of the ToT strategy alone, with an emphasis on the capabilities of the generator and discriminator or the complexity of the task. Chen et al. (2024) explored the conditions under which tree search methods can enhance the performance of LLMs in planning tasks. Through empirical studies, the authors concluded that the effectiveness of tree search heavily depends on the discriminator's ability to accurately evaluate and guide the search process. Notably, their findings were based on the use of a single generator (i.e., CodeLlama-13B-Inst (Roziere et al., 2023)). In contrast, our work employs three different scales of LLMs and an accuracy-controlled oracle generator, providing a more comprehensive investigation into the impact of both the generator and discriminator on the overall performance of the ToT framework.

## 6 Conclusion

This study demonstrates that while ToT offers theoretical advantages, its practical benefits are realized only under specific conditions — namely, when both the generator and discriminator are sufficiently capable. Our findings show that ToT can significantly improve LLM reasoning abilities, but this enhancement depends primarily on the quality of the generator. In tasks with large search spaces, such as Game of 24, stronger generators lead to higher success rates. While the accuracy of the discriminator also contributes to ToT's performance, stronger LLMs do not provide superior discrimination performance. Therefore, we recommend using a state-of-the-art large model as the generator alongside a smaller model as the discriminator to harness the advantages of ToT while significantly reducing computational costs, without the need for extreme performance optimization. Furthermore, the benefits of ToT are more pronounced in highly complex tasks where methods like IO and CoT show poor performance, emphasizing the value of using ToT in such challenging scenarios.

## 7 Limitations

This study has several limitations. First, we only experimented with the Knights and Knaves subset containing three characters, which led to overly optimistic performance for the baseline methods, partially masking the advantages of ToT. Second, the reliance on oracle modules to control accuracy limits the applicability of our findings to real-world scenarios, as it fails to fully capture the inherent variability in LLM performance. Additionally, the parameter settings for ToT are another important factor influencing its effectiveness. Future work should expand the range and complexity of tasks and consider the impact of ToT's parameter settings to gain a more comprehensive understanding of its effectiveness.

## References


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

AI@Meta. 2024. Llama 3 model card.

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

Maciej Besta, Florim Memedi, Zhenyu Zhang, Robert Gerstenberger, Guangyuan Piao, Nils Blach, Piotr Nyczyk, Marcin Copik, Grzegorz Kwa'sniewski, Jurgen Muller, Lukas Gianinazzi, Ale Kubíek, Hubert Niewiadomski, Aidan O'Mahony, Onur Mutlu, and Torsten Hoefler. 2024b. Demystifying chains, trees, and graphs of thoughts.

Shulin Cao, Jiajie Zhang, Jiaxin Shi, Xin Lv, Zijun Yao, Qi Tian, Lei Hou, and Juanzi Li. 2023. Probabilistic tree-of-thought reasoning for answering knowledge-intensive complex questions. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 12541–12560, Singapore. Association for Computational Linguistics.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*.

Ziru Chen, Michael White, Raymond Mooney, Ali Payani, Yu Su, and Huan Sun. 2024. When is tree search useful for llm planning? it depends on the discriminator. In *Annual Meeting of the Association for Computational Linguistics*.

Ruomeng Ding, Chaoyun Zhang, Lu Wang, Yong Xu, Minghua Ma, Wei Zhang, Si Qin, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. 2024. Everything of thoughts: Defying the law of penrose triangle for thought generation. *Preprint*, arXiv:2311.04254.

Jinhao Duan, Renming Zhang, James Diffenderfer, Bhavya Kailkhura, Lichao Sun, Elias Stengel-Eskin, Mohit Bansal, Tianlong Chen, and Kaidi Xu. 2024. Gtbench: Uncovering the strategic reasoning limitations of llms via game-theoretic evaluations. *CoRR*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly,

Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaoqing Ellen Tan, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aaron Grattafiori, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alex Vaughan, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Franco, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, Danny Wyatt, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Firat Ozgenel, Francesco Caggioni, Francisco Guzmán, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Govind Thattai, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Karthik Prasad, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kun Huang, Kunal Chawla, Kushal Lakhotia, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Maria Tsimpoukelli, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikolay Pavlovich Laptev, Ning Dong, Ning Zhang, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Rohan Maheswari, Russ Howes, Ruty Rinott, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Kohler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vítor Albiero, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaofang Wang, Xiaojian Wu, Xiaolan Wang, Xide Xia, Xilun Wu, Xinbo Gao, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yuchen Hao, Yundi Qian, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, and Zhiwei Zhao. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks. In *The Eleventh International Conference on Learning Representations*.

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.

Shentong Mo and Miao Xin. 2024. Tree of uncertain thoughts reasoning for large language models. In *ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 12742–12746.

Philipp Mondorf and Barbara Plank. 2024. Liar, liar, logical mire: A benchmark for suppositional reasoning in large language models. *Preprint*, arXiv:2406.12546.

Edward F Moore. 1959. The shortest path through a maze. In *Proc. of the International Symposium on the Theory of Switching*, pages 285–292. Harvard University Press.

Maxwell Nye, Anders Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. 2021. Show your work: Scratchpads for intermediate computation with language models. *ArXiv*, abs/2112.00114.

Santosh Kumar Radha, Yasamin Nouri Jelyani, Ara Ghukasyan, and Oktay Goktas. 2024. Iteration of thought: Leveraging inner dialogue for autonomous large language model reasoning. *Preprint*, arXiv:2409.12618.

Leonardo Ranaldi, Giulia Pucci, Federico Ranaldi, Elena Sofia Ruzzetti, and Fabio Massimo Zanzotto. 2024. Empowering multi-step reasoning across languages via tree-of-thoughts. *Preprint*, arXiv:2311.08097.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Bilgehan Sel, Ahmad Tawaha, Vanshaj Khattar, Ruoxi Jia, and Ming Jin. Algorithm of thoughts: Enhancing exploration of ideas in large language models. In *Forty-first International Conference on Machine Learning*.

Vered Shwartz, Peter West, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2020. Unsupervised commonsense question answering with self-talk. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4615–4629, Online. Association for Computational Linguistics.

Robert Tarjan. 1972. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160.

Gemma Team. 2024. Gemma.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, Pouya Tafti, Léonard Hussenot, Pier Giuseppe Sessa, Aakanksha Chowdhery, Adam Roberts, Aditya Barua, Alex Botev, Alex Castro-Ros, Ambrose Slone, Amélie Héliou, Andrea Tacchetti, Anna Bulanova, Antonia Paterson, Beth Tsai, Bobak Shahriari, Charline Le Lan, Christopher A. Choquette-Choo, Clément Crepy, Daniel Cer, Daphne Ippolito, David Reid, Elena Buchatskaya, Eric Ni, Eric Noland, Geng Yan, George Tucker, George-Christian Muraru, Grigory Rozhdestvenskiy, Henryk Michalewski, Ian Tenney, Ivan Grishchenko, Jacob Austin, James Keeling, Jane Labanowski, Jean-Baptiste Lespiau, Jeff Stanway, Jenny Brennan, Jeremy Chen, Johan Ferret, Justin Chiu, Justin Mao-Jones, Katherine Lee, Kathy Yu, Katie Millican, Lars Lowe Sjoesund, Lisa Lee, Lucas Dixon, Machel Reid, Maciej Mikuła, Mateo Wirth, Michael Sharman, Nikolai Chinaev, Nithum Thain, Olivier Bachem, Oscar Chang, Oscar Wahltinez, Paige Bailey, Paul Michel, Petko Yotov, Rahma Chaabouni, Ramona Comanescu, Reena Jana, Rohan Anil, Ross McIlroy, Ruibo Liu, Ryan Mullins, Samuel L Smith, Sebastian Borgeaud, Sertan Girgin, Sholto Douglas, Shree Pandya, Siamak Shakeri, Soham De, Ted Klimenko, Tom Hennigan, Vlad Feinberg, Wojciech Stokowiec, Yu hui Chen, Zafarali Ahmed, Zhitao Gong, Tris Warkentin, Ludovic Peran, Minh Giang, Clément Farabet, Oriol Vinyals, Jeff Dean, Koray Kavukcuoglu, Demis Hassabis, Zoubin Ghahramani, Douglas Eck, Joelle Barral, Fernando Pereira, Eli Collins, Armand Joulin, Noah Fiedel, Evan Senter, Alek Andreev, and Kathleen Kenealy. 2024. Gemma: Open models based on gemini research and technology. *Preprint*, arXiv:2403.08295.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023a. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2609–2634.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023b. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc.

Yilin Wen, Zifeng Wang, and Jimeng Sun. 2024. Mindmap: Knowledge graph prompting sparks graph of thoughts in large language models. *Preprint*, arXiv:2308.09729.

Siwei Yang, Bingchen Zhao, and Cihang Xie. 2024. Aqa-bench: An interactive benchmark for evalu-

ating llms' sequential reasoning ability. *Preprint*, arXiv:2402.09404.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 11809–11822. Curran Associates, Inc.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V Le, et al. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations*.

# A Experimental Setup

## A.1 Baselines & ToT Setup

### A.1.1 Game of 24

Our implementation is largely based on Yao et al. (2023)'s work. However, during our pilot experiment, we found that the same setup did not allow the selected open-source model to understand the task well. Therefore, we made some adjustments based on (Yao et al., 2023)'s work.

In terms of IO prompting, we follow (Yao et al., 2023)'s approach of using five in-context examples to support the IO prompt, but we adjust the order of the examples. The IO prompt is as follows:

**IO Prompt**

```
Use numbers and basic arithmetic operations (+ - * /)
to obtain 24.
Input: 2 9 10 12
Answer: 2 * 12 * (10 - 9) = 24
Input: 4 9 10 13
Answer: (13 - 9) * (10 - 4) = 24
Input: 1 4 8 8
Answer: (8 / 4 + 1) * 8 = 24
Input: 5 5 5 9
Answer: 5 + 5 + 5 + 9 = 24
Input: 4 4 6 8
Answer: (4 + 8) * (6 - 4) = 24
Input: <input>
```

As for CoT, we use the same prompt as (Yao et al., 2023), add three intermediate equations in each input-output pair, and present five examples to LLMs. The CoT prompt is as follows:

**CoT Prompt**

```
Use numbers and basic arithmetic operations (+ - *
/) to obtain 24. Each step, you are only allowed to
choose two of the remaining numbers to obtain a new
number.
Input: 4 4 6 8
Steps:
4 + 8 = 12 (left: 4 6 12)
6 - 4 = 2 (left: 2 12)
2 * 12 = 24 (left: 24)
Answer: (6 - 4) * (4 + 8) = 24
Input: 2 9 10 12
Steps:
12 * 2 = 24 (left: 9 10 24)
10 - 9 = 1 (left: 1 24)
24 * 1 = 24 (left: 24)
Answer: (12 * 2) * (10 - 9) = 24
Input: 4 9 10 13
Steps:
13 - 10 = 3 (left: 3 4 9)
9 - 3 = 6 (left: 4 6)
4 * 6 = 24 (left: 24)
Answer: 4 * (9 - (13 - 10)) = 24
Input: 1 4 8 8
Steps:
8 / 4 = 2 (left: 1 2 8)
1 + 2 = 3 (left: 3 8)
3 * 8 = 24 (left: 24)
Answer: (1 + 8 / 4) * 8 = 24
Input: 5 5 5 9
Steps:
5 + 5 = 10 (left: 5 9 10)
10 + 5 = 15 (left: 9 15)
15 + 9 = 24 (left: 24)
Answer: ((5 + 5) + 5) + 9 = 24
Input: <input>
```

Regarding the ToT approach, we emulate the steps of the CoT process. In each step, the generator produces multiple candidate explanations. The discriminator then evaluates each candidate explanation three times and selects the top five to proceed to the next step. Consequently, the LLM generates up to five answers for each task. For fairness, we also instruct the LLM to generate five answers when using the baseline methods. To mitigate the inflated success rate from multiple attempts, we use average success rate (see Section 3.2) alongside overall success rate.

In pilot experiments, we observe that some small open-source models struggled to generate valid intermediate steps, resulting in inefficient ToT performance and poor experimental outcomes. To address this, we introduce a filter in the generation step to discard obviously invalid intermediate steps (e.g., steps with mismatched remaining numbers), before passing the filtered candidates to the discriminator for evaluation. This not only enhancs ToT efficiency but also improves task success rates to a certain extent.

For the generator, we use a three-shot generation prompt to produce equations for the first three steps, and a five-shot merge prompt to consolidate the intermediate steps into a single equation in the final step. For the discriminator, we similarly employ a

three-shot value prompt to guide LLMs in assessing the potential of completing the game in the first three steps, and another zero-shot value prompt to evaluate the correctness of the final step. The four prompts are as follows.

**ToT: Generation Prompt**

Input: 2 8 8 14
Possible next steps:
2 + 8 = 10 (left: 8 10 14)
8 / 2 = 4 (left: 4 8 14)
14 + 2 = 16 (left: 8 8 16)
2 * 8 = 16 (left: 8 14 16)
8 - 2 = 6 (left: 6 8 14)
14 - 8 = 6 (left: 2 6 8)
14 / 2 = 7 (left: 7 8 8)
14 - 2 = 12 (left: 8 8 12)
Input: 4 4 10
Possible next steps:
4 + 4 = 8 (left: 8 10)
4 * 10 = 40 (left: 4 40)
10 - 4 = 6 (left: 4 6)
4 / 4 = 1 (left: 1 10)
4 - 4 = 0 (left: 0 10)
4 + 10 = 14 (left: 4 14)
4 * 4 = 16 (left: 10 16)
Input: 10 14
Possible next steps:
10 + 14 = 24 (left: 24)
14 - 10 = 4 (left: 4)
10 * 14 = 140 (left: 140)
14 / 10 = 1.4 (left: 1.4)
Generate possible next steps for the following inputs, following the example above. Note that the number of leftover digits should be one less than the number of input digits.
Input: <input>
Possible next steps:

**ToT: Merge Prompt**

Given three calculation steps. Follow the examples and combine the three calculation steps into one equation, but do not simplify. Your output should be in this format "Answer: {combined one equation}"
Examples:
Steps:
8 / 4 = 2 (left: 1 2 8)
1 + 2 = 3 (left: 3 8)
3 * 8 = 24 (left: 24)
Answer: (1 + 8 / 4) * 8 = 24
Steps:
12 * 2 = 24 (left: 9 10 24)
10 - 9 = 1 (left: 1 24)
24 * 1 = 24 (left: 24)
Answer: (12 * 2) * (10 - 9) = 24
Steps:
4 + 8 = 12 (left: 4 6 12)
6 - 4 = 2 (left: 2 12)
2 * 12 = 24 (left: 24)
Answer: (6 - 4) * (4 + 8) = 24
Steps:
5 + 5 = 10 (left: 5 9 10)
10 + 5 = 15 (left: 9 15)
15 + 9 = 24 (left: 24)
Answer: ((5 + 5) + 5) + 9 = 24
Steps:
13 - 10 = 3 (left: 3 4 9)
9 - 3 = 6 (left: 4 6)
4 * 6 = 24 (left: 24)
Answer: 4 * (9 - (13 - 10)) = 24
It's your turn:
Steps: <steps>

**ToT: Value Prompt**

Evaluate if given numbers can reach 24. Conclude in the last line "confident", "likely" or "impossible".
Example 1:
4 4 10
4 + 4 + 10 = 8 + 10 = 18
4 * 10 - 4 = 40 - 4 = 36
(10 - 4) * 4 = 6 * 4 = 24
confident
Example 2:
11 12
11 + 12 = 23
12 - 11 = 1
11 * 12 = 132
11 / 12 = 0.91
impossible
Example 3:
5 7 8
5 + 7 + 8 = 12 + 8 = 20
(8 - 5) * 7 = 3 * 7 = 21
I cannot obtain 24 now, but numbers are within a reasonable range
likely
Now you should evaluate:
<input>

**ToT: Value Prompt for last step**

Given an input and an answer, evaluate if the answer is correct, i.e. it uses each input exactly once and no other numbers, calculation is correct and reaches 24.Give your judgement in the last line: "confident" or "impossible".
Input: <input>
Answer: <answer>

In the experiments described in Sections 4.1 and 4.2, we utilize a dataset containing 1,000 tasks, while in the experiment detailed in Section 4.3, we employ the complete dataset of 1,362 tasks.

### A.1.2 Knights and Knaves

This study adopts Mondorf and Plank (2024)'s zero-shot IO- and CoT prompts as baselines. The IO prompt consists of a system message and an instruction, while the CoT prompt extends the IO prompt by adding "Let's think step by step."

**System Message**

Your task is to solve a logical reasoning problem. You are given set of statements from which you must logically deduce the identity of a set of characters.

You must infer the identity of each character. First, explain your reasoning. At the end of your answer, you must clearly state the identity of each character by following the format:

CONCLUSION:
A: ...
B: ...
C: ...
...

**Instruction**

### Instruction ###
Assume that there exist only two types of people: truth-tellers and liars. truth-tellers always tell the truth, while liars always lie.
You are given the statements from <num-characters> characters. Based on their statements, infer who is a truth-teller and who is a liar.

Based on the following statements, infer who is a truth-teller and who is a liar:
<statements>

First, explain your reasoning. End your answer by clearly stating the identity of each character in the following format:

A: truth-teller/liar
B: truth-teller/liar
C: truth-teller/liar
...

In the ToT setup, we have the LLM agent analyze each character's identity step by step, where reasoning at each subsequent step is based on inferences from prior steps. In this way, the Knights and Knaves problem can be viewed as a full binary tree with a depth equal to the number of characters. The two nodes at each level represent the two possible identities of a character (truth-teller/liar), and the unique solution corresponds to one root-to-leaf path. Since each character has only two possible identities, the default number of generated explanations is set to 2. The discriminator uses a three-round voting mechanism to select the best explanation from the two candidates. In practice, LLMs allows the generation of two explanations with the same conclusion. The prompt for Knights and Knaves is as follows, we make it zero-shot to be consistent with baseline setup:

**ToT: Generation Prompt**

### Instruction ###
Assume that there exist only two types of people: truth-tellers and liars. truth-tellers always tell the truth, while liars always lie.
You are given the statements from <num-characters> characters. Based on their statements, and some known identities, infer who is a truth-teller and who is a liar.

Statements:
<statements>

Known identities:
<known_identities>

Now, infer the identity of <character> and explain your reasoning. End your answer by clearly stating the identity of <character> in the following format:

<character>: truth-teller/liar

**ToT: Vote Prompt**

Given an instruction, several statements, known identities and several choices, decide which choice is most promising.
Analyze each choice in detail, then conclude in the last line "The best choice is {s}", where s the integer id of the choice.

### Instruction ###
Assume that there exist only two types of people: truth-tellers and liars. truth-tellers always tell the truth, while liars always lie.
You are given the statements from <num-characters> characters. Based on their statements, and some known identities, infer who is a truth-teller and who is a liar.

Statements:
<statements>

Known identities:
<known_identities>

Choice 1: <first candidate explanation>
Choice 2: <second candidate explanation>

Unless otherwise specified, we use the subset of the Knights and Knaves task with 3 characters.

### A.2 Hardware

Our experiments are conducted on NVIDIA-A100 GPUs, each with either 40 GB or 80 GB of memory.

## B Theoretical Calculation of the Branching Factor in the Game of 24 Game Tree

1. In the first step, players can select two numbers from four, and choose one from the four

basic arithmetic operators to perform calculations. Since subtraction and division do not obey the commutative law, if we consider equivalent operations that obey the commutative law as different choices, then the exploration space for the first step is $4 \times 3 \times \binom{4}{1} = 48$.

2. In the second step, players can select two numbers from the remaining three, and choose one from the four basic arithmetic operators to perform calculations. Therefore, the exploration space for the second step is $3 \times 2 \times \binom{4}{1} = 24$.

3. In the third step, only two numbers remain. Players need to choose one from the four basic arithmetic operators to perform calculations. Therefore, the exploration space for the third step is $2 \times 1 \times \binom{4}{1} = 8$.

4. In the fourth step, players only need to integrate the previous three calculation steps into one formula. We assume that the player combines the three expressions based on the sequence of the first three intermediate steps, without considering the commutative property or equivalent operations using parentheses. Therefore, the exploration space for the fourth step is 1.

In summary, the branching factor for Game of 24 is {1, 8, 24, 48}.

It is important to note that our calculations are based on the assumption that the four initial numbers are completely distinct (e.g., 1, 2, 3, 4). If there are repetitions among the four numbers (e.g., 1, 8, 8, 8, or 6, 6, 6, 6), the actual branching factor would be smaller. The specific branching factor depends on the nature of the repetitions. Given that the repetition of remaining numbers in the intermediate steps largely depends on the calculations considered by the player, we simplify our computation by assuming all numbers are distinct. Consequently, the branching factor we obtain represents the upper limit for all possible scenarios.

## C Algorithm for generating complete answers to Game of 24

**Algorithm 1** Solve 24 Game

```
function SOLVE_24_GAME(numbers, target = 24)
    if numbers is empty then
        return empty list
    end if
    Initialize solutions as empty list
    Set ϵ to small tolerance value for floating point comparison
    function DFS(nums, steps)
        if length of nums = 1 then
            if nums[0] is approximately target within tolerance ϵ then
                Add current steps to solutions
            end if
            return
        end if
        for each pair (a, b) from nums do
            if a = b then
                continue
            end if
            Create next_nums by removing a and b from nums
            Define operations list as:
            a + b, a − b, b − a, a ∗ b
            if b ≠ 0 then
                Add a/b to operations
            end if
            if a ≠ 0 then
                Add b/a to operations
            end if
            for each operation result, expression in operations do
                Append expression to steps
                Call dfs(next_nums + result, updated steps)
            end for
        end for
    end function
    Call dfs(numbers, empty steps)
    return solutions
end function
```