Evaluation of the Tree of Thoughts Prompting Approach for Large Language Models based on the problem of Optimal Binary Search Trees

Kilian Hüllen[0000−0001−9411−8146]

University of Mannheim, Mannheim, Germany
kilian.huellen@students.uni-mannheim.de

Abstract. Large Language Models (LLMs) have gained significant popularity in recent years as general problem solvers. Besides research, developments and enhancements on the Language Models themselves, prompt engineering and different prompting techniques have been catching the attention of researchers and businesses as they have proven to be capable of increasing the accuracy and correctness of LLM responses significantly. One of these techniques is the "Tree of Thoughts" (ToT) approach that showed promising results in past experiments with tasks of limited complexity. ToT is based on an algorithm guiding through a tree-like structure of single thoughts, i.e., intermediate and partial solutions, that lead to a final response by the LLM. The experiments presented and evaluated in this paper try to solve the problem of creating Optimal Binary Search Trees (OBSTs) with minimal search costs based on key values and their respective frequencies. The experiment results clearly show the limitations of the ToT framework: While the approach still slightly improves the results in comparison to other common prompting techniques, and especially compared to a zero-shot prompt, the impressive improvements seen for other problems do not materialize for the problem of creating OBSTs. Additionally, a cost analysis shows the high expenses in the communication with the LLM leading to the conclusion that for using ToT rather than other prompting techniques, the better results must justify the increased cost and time investments for the exchange with the LLM in every single use case. Complete code repository can be found on: https://github.com/kilian1322/tot-obst

Keywords: Tree of Thoughts, Optimal Binary Search Trees, Large Language Models, Chain of Thought, Zero-Shot, Prompting Techniques.

1 Introduction

Originally developed for natural language understanding and text generation, Large Language Models (LLMs) have been able to prove their increasingly growing areas of application in recent years [3, 16]. Current models are able to provide valuable outputs in amongst others, but not limited to, the areas of code generation and refactoring, financial analyses, education and scientific research, healthcare and many more [11]. Additionally, these models strive to consistently deliver

correct results in mathematical and logical reasoning as well as in increasingly complex tasks in general [2, 4, 7, 17].

With this wide range of problem solving capabilities, LLMs have become a promising candidate for general problem solvers. However, the capabilities of LLMs themselves do not determine the quality of their outputs alone. Rather, the prompts taken as input from the LLM pose a significant factor as well. Therefore, prompt engineering has become a common practice and significant research field to further extend the set of problems and use cases for which LLMs deliver valuable solutions.

Prompt engineering can be as simple as optimising the comprehensibility and syntax of an individual prompt, using specific key words that help the model sharpening its response, or by estimating a trade-off between too short and unspecific prompts and too long and exhausting ones. However, this basic kind of prompt engineering only covers the idea of Input-Output Prompting where one prompt is being optimised to directly request the final answer from the LLM.

More complex approaches, which also include the ToT Approach, are based on the idea that the problem solving process can rather be seen as an interactive dialogue between the LLM and the agent creating prompts and evaluating the responses, than just a single input-output interaction. Throughout this dialogue, the LLM can form different intermediate thoughts that consecutively lead to a final answer. How to construct this interactive process the best is the central question that the Chain of Though (CoT), ToT and Graph of Thoughts (GoT) Approaches deal with.

1.1 Towards ToT: Different Prompting Methods

ToT, together with its underlying algorithm, emerged from previously developed approaches. For a deeper understanding of ToT, the approach is explained here together with its related prompting techniques.

Fig. 1 [1] illustrates the differences between the basic Input-Output (IO), the CoT and ToT Prompting. GoT [1, 5, 15] is an even more complex prompting algorithm that is based on the idea of ToT. Therefore, GoT is not needed to gain an understanding of the ToT approach and is not covered in more depth in this paper.

The IO Prompting is the most simple and intuitive way to gain responses from an LLM. A prompt containing the question or problem and, either explicitly or implicitly, asking for an appropriate answer or solution, is being sent to the Language Model as the input, and the response of the model is taken as the output. Thereby, the single prompt that is being issued contains the complete problem statement, and the response must be seen as the final solution created by the language model.

The CoT Approach is the first one to take advantage of a more complex solution-building interaction between the requesting agent (e.g., a human) and the LLM. It deals with different "Thoughts" and uses them as intermediate steps

[1] extracted from [14]

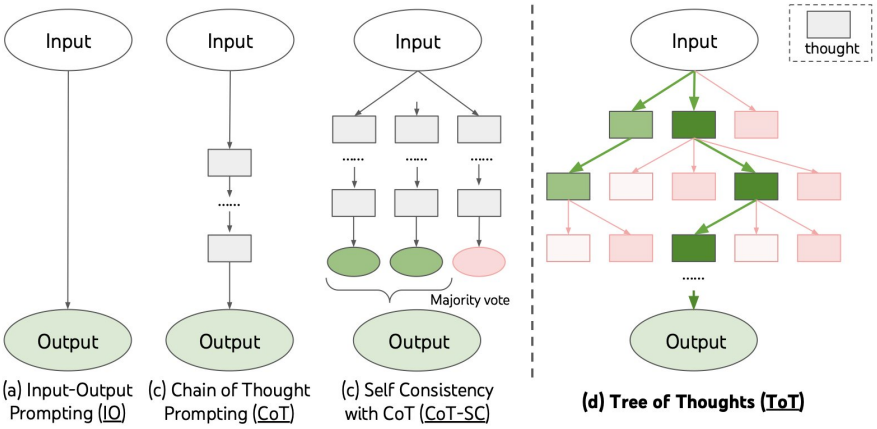


Fig. 1. ToT Approach visually explained

towards the final solution. In the context of prompt engineering, a thought is one coherent sequence of words or characters that has a semantic meaning in the realm of the given problem [14]. For CoT, the questioning agent asks the LLM to create multiple thoughts step by step that lead to the final solution. It has been shown that the breakdown of a problem into smaller sub-problems and the incentive for the model to explore the solution space in multiple steps increases the correctness and quality of final responses significantly [10, 13].

A further development of CoT is Self Consistency with CoT (CoT-SC) that relies completely on the concept and implementation of CoT but scales up the number of parallel conversations between the agent and the language model. Each one of these conversations starts with the original question or problem and represent one complete CoT cycle. This results in several final answers from the LLM, and the most promising one can be picked as the output to the initial problem. Deciding over the most promising answer can either be done manually by a human or an external agent, or by the LLM itself - either by a majority vote over all final answers, or by letting the model assess the most promising answer itself. This scaled up version of CoT of course linearly increases the costs of the prompting technique based on the number of parallel CoT runs, but it also showed improvements in the correctness and quality of the final output compared to IO and CoT Prompting [12].

While the step from CoT to CoT-SC simply introduces the idea of self-consistency across different final solutions but still heavily relies on the overall idea and implementation of CoT, the ToT approach significantly extends CoT and also introduces a new underlying data structure. Instead of building a chain of thought that strictly leads towards the final solution, ToT builds a whole tree of different thoughts and navigates through this tree until eventually reaching a leaf node that is taken as the final answer and output of the LLM. As

the underlying data structure is a tree, more advanced operations are needed in the implementing algorithm, and the costs are likely to become even higher than the costs of CoT-SC. However, in previously conducted experiments, ToT performed significantly better than any of the other introduces prompting techniques. Especially at problems that seem to be just too complex for CoT to deliver satisfactory results, ToT still performs very well [6, 9, 14].

1.2 Scope of this paper

However, the problems from previous experiments are still of very limited complexity and easily solvable by humans. Thus, with the problem of creating Optimal Binary Search Trees (OBSTs), this paper presents the experiments and evaluations of a more complex problem design for a broader understanding of the capabilities and limitations of ToT.

First, we will review the experiments conducted so far and their results to better evaluate ToT on a global scale in the end. Then, the experimental design of solving the problem of OBSTs with ToT will be described in detail, including the implemented algorithm and the different prompts that are sent to the LLM in the process. Additionally, the concept of OBSTs and their construction will be covered briefly to gain an understanding of the complexity of this problem and possible challenges when solving it with LLMs. Lastly, the experimental results will be presented and discussed as well as an error and a cost analysis, ultimately resulting in a broader understanding of the opportunities and limitations of ToT.

2 Related Work

The ToT prompting approach was first introduced in 2023 by Long, and Yao et al. [6, 14]. Additionally, Ranaldi and Zanzetto [9] first discovered the application of ToT across multiple natural languages. In their works, they all refer to the already established CoT approach as the baseline for their conceptual structure of ToT.

As the ideas and algorithms presented by Yao et al. and Long serve as a baseline for the experiments conducted in this paper, in the following their experimental setups and results will be discussed briefly.

Long [6] introduces a whole ToT software system composed of four different components for communicating with the LLM and controlling the exploration of thoughts inside the tree of thoughts. These components are a checker module to ensure the LLM responses meet the necessary syntactic and semantic requirements, a memory module to keep track of the tree that has been explored so far, a prompter agent that sends the suitable prompts to the LLM, and the ToT controller that communicates with the memory module and the prompter agent to ensure a flawless and optimal walk through the tree of thoughts.

Using this structure and an underlying ToT algorithm for the ToT controller, Long tries to solve Sudoku puzzles of the sizes 3x3, 4x4 and 5x5 and compares the results of their introduced approach with the results of 0-shot, 1-shot and CoT

prompting. Fig. 2 [2] shows the results of the experiments. ToT clearly outperforms each of the other prompting techniques, and especially for larger Sudokus, i.e., when the problem that needs to be solved gets increasingly more complex, ToT is the only approach that still provides a comparatively high reliability.

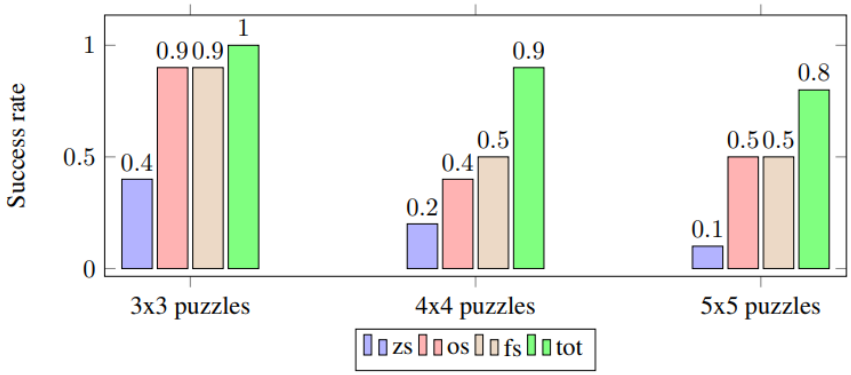


Fig. 2. Results of Sudoku puzzle solver with 0-shot (zs), 1-shot (os), CoT (fs) and ToT approaches

The second publication by Yao et al. [14] does not rely on a strict separation of responsibilities in different modules. Rather, they provide an all-encompassing ToT algorithm managing the tree of thoughts exploration. The major phases during this procedure are the thought generation and state evaluation. Thereby, in every step of the algorithm, a new set of propositional thoughts is created and evaluated by the LLM, and the most promising thoughts are kept for the following rounds of the search.

With this approach, Yao et al. conduct a number of different experiments: the Game of 24, Creative Writing and 5x5 crossword puzzles.

The Game of 24 takes four natural numbers as an input with the goal to reach the number 24 with basic mathematical operations ($+$, -, $\times$, $\div$).

For Creative Writing, four random sentences were given with the task to write a coherent passage of four paragraphs respectively ending in the four provided sentences. The intermediate thought step is introduced by asking the model to first develop different writing plans and then choosing the most promising one to write the text passage.

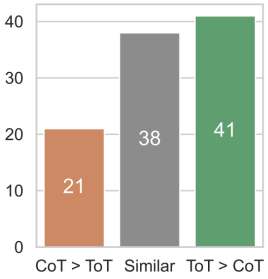
And lastly, the 5x5 crossword puzzle takes ten hints as input where each hint relates to one row or column of the final crossword puzzle. The intermediate thoughts consist of possible words the LLM comes up with to fill the crossword puzzle with. When the evaluation of these proposals reach an adequate level of certainty, the words are set to represent a part of the final solution.

[2] extracted from [6]

Once again, the results of these three experiments show a clear dominance of ToT as it outperforms IO, CoT and CoT-SC prompting for all tested problems as seen in Fig. 3 [3].

Method	Success
IO prompt	7.3%
CoT prompt	4.0%
CoT-SC (k=100)	9.0%
ToT (ours) (b=1)	45%
ToT (ours) (b=5)	**74%**
IO + Refine (k=10)	27%
IO (best of 100)	33%
CoT (best of 100)	49%

(a) Game of 24



(b) Creative Writing. Human evaluation on which passage is more coherent

Method	Success Rate (%)		
	Letter	Word	Game
IO	38.7	14	0
CoT	40.6	15.6	1
ToT (ours)	**78**	**60**	**20**
+best state	82.4	67.5	35
-prune	65.4	41.5	5
-backtrack	54.6	20	5

(c) Mini Crossword Puzzles

Fig. 3. Results for the experiments conducted by Yao et al.

However, it is also noticeable that in some situations, ToT seems to reach its limits as well. For example, while ToT still performs well in finding single words in a crossword (60% success rate), only 20% of crossword puzzles are solved correctly as a whole.

3 Experimental Setup

3.1 The Problem of Optimal Binary Search Trees

The creation of Optimal Binary Search Trees is applied in amongst others the areas of database systems, compiler optimization and data compression through techniques like Huffman coding [8]. Additionally, it is a problem that is still very well solvable by humans but in the meantime more difficult than problems conducted in previous papers discussing the ToT approach. Therefore, it is well suited to further examine the capabilities and limitations of LLMs and ToT in solving specific problems of different difficulties.

The problem of OBST construction takes a list of key values keys[1, ..., n] and their respective frequencies freq[1, ..., n] as input parameters. The key values can be of any set that defines a total order, and the frequencies can be either absolute or relative values. If the frequencies are expressed relatively, they add up to 1, i.e., 100%, and one key-frequency pair can be understood as follows: The probability that the key value keys[i] is being searched for is freq[i].

When a binary tree structure is used to search for the different keys, OBSTs minimize the expected search time based on the known search frequencies.

[3] extracted from [14]

For a Binary Search Tree (BST), the expected search time for a key can be calculated as follows:

$$\frac{\sum_{k \in keys}(level(k) \cdot freq(k))}{|keys|} \tag{1}$$

where the level of a value k refers to the level inside the binary tree of the node encapsulating k.

The most commonly used way to compute the OBST for a given keys and frequencies pair is through Dynamic Programming and a Bottom-Up approach: A table of OBSTs is being constructed where in the last step, the whole tree over all $|keys|$ nodes is created. In each step until then, only a partial OBST over a subset of all keys is calculated. Starting with only one node, in every further step the creation of yet another OBST is guaranteed by calculating the expected search time for all nodes as head node of the new tree, only keeping track of the optimal partial solution to build on in the next iteration.

3.2 Data Set used for the experiment

For the experiments conducted in this paper, a data set of 20 keys-frequencies pairs were created, splitting into five instances each with 3, 5, 7 and 9 keys each. The pairs were handcrafted but it was ensured that the resulting OBSTs cover a variety of different use cases, frequency weightings and thus tree structures. Table 1 provides an overview over the distribution of the depths of the resulting OBSTs from the data set.

depth	2	3	4	5
frequency	2	7	9	2

Table 1. Frequencies of resulting depths of the 20 OBSTs

3.3 Underlying ToT algorithm

For a given keys-frequencies pair as the input, the implemented ToT algorithm manages the communication with the LLM and the exploration through the spanned tree of different thoughts. The LLM used is the latest version of GPT 3.5 at the time of the experiments, i.e., gpt-3.5-turbo-1106, with the temperature parameter set to 0.7. Further technical specifications can be seen in the GitHub repository [4].

Algorithm 1 provides a general overview of the procedure of the underlying ToT algorithm with a focus on the interactions with the LLM. For a successful guidance through the different thoughts extracted from the LLM, in every thought step, three chronological steps are of high importance: First, based on

[4] https://github.com/kilian1322/tot-obst

Algorithm 1 ToT

Input: keys, freq, maxRounds

$step \leftarrow 0$
$thoughts \leftarrow [\,]$
$solutions \leftarrow [\,]$
while $step \neq maxRounds$ **do**
  $proposals \leftarrow getProposals(keys, freq, thoughts)$
  $thoughts \leftarrow evalueteProposals(keys, freq, proposals, thoughts)$
  for *thought* `in` *thoughts* **do**
    $solutions \leftarrow checkCompleteness(thought)$
  end for
end while
$finalObst \leftarrow getBestSolution(solutions)$

Output: finalObst

the intermediate results obtained so far, the language model needs to create multiple proposals for a subsequent thought. Next, the quality of every single of these newly created thoughts needs to be estimated in regards to the likelihood to lead towards a correct output. And lastly, for all thoughts that have been rated to be good enough to be used as the baseline in the next loop of the algorithm, it must be considered whether they already provide a final solution and therefore do not need to be taken into account as intermediate thoughts anymore.

This last step was not necessary for the problems discussed in previous papers on ToT as all these problems always had the same number of intermediate thoughts for all instances of one problem type. However, as different BSTs - even with the same number of key values - can have completely different structures and number of levels, they do not have one unique number of intermediate steps and thoughts. Therefore, for every partial solution, the ToT algorithm needs to check whether this partial solution already represents a complete BST with all the key values in it. This is then considered as a final solution to the given problem of creating an OBST.

In the following, these three key steps of the algorithm, together with their respective prompts that are sent to the LLM, will be discussed in further detail.

Propose Prompt To further span our tree of thoughts in the process of problem solving, the language model is asked to create different proposals for the subsequent thought based on the most promising thoughts from the last round. This is done using the following prompt:

> Given an array k and an array freq where k contains key values and freq the respective frequencies for the use of the keys. The goal is to construct an optimal binary search tree to minimize the average search time for the key values. In the very end, output the array representing the optimal

> binary search tree. [... As an intermediate step], propose possible next steps in the construction of the optimal binary search tree [...].
>
> Input: k=[1,2,3,4,5], freq=[11,8,6,10,4], and the partially created tree [2].
> Possible next steps in the Optimal Binary Tree Construction:
> [2, 1, 3], [2, 1, 4], [2, 1, 5], [2, null, 3], [2, null, 4], [2, null, 5], [2, 1, null]
>
> Input: {input}
> Possible next steps in the Optimal Binary Tree Construction:
> Your proposals here

The proposals for the intermediate binary trees obtained from the LLM should be in the format of an array representing the respective binary tree. The specific formulation of the prompt ensures that the model only returns a listing of arrays which is needed for the further automatic processing of the LLM response.

Value Prompt After obtaining different proposals from the language model, these new intermediate thoughts need to be evaluated to decide whether they should be kept and explored further. Therefore, the set of all currently promising thoughts together with the new proposals is being evaluated. Using the Value Prompt, every thought and proposal is taken individually and the LLM is asked for an evaluation of the likelyhood of this thought to lead towards the correct final solution:

> Given a partially constructed binary tree, evaluate whether it has a chance of becoming the Optimal Binary Search Tree and how high that chance is. Give an estimator value representing this chance. The estimator value must be an integer between 1 and 10 where 1 indicates a very low chance, and 10 a very high chance. [...] Input: {input}
>
> Reasoning:
> Judgement: {Fill in your estimator value here. Value must be an integer between 1 and 10, and one specific value MUST be provided.}

Based on these evaluations, the most promising thoughts are being kept for further exploration. Depending on how many thoughts are kept, the spanned tree of thoughts gets respectively wider or stays rather narrow. In our experiments, we specified to keep track of the five most promising thoughts.

Completeness Prompt When the five most promising intermediate thoughts have been chosen, it must be made sure for all of them that they do not yet represent a complete BST already containing all key values. Therefore, for all five intermediate thoughts that is kept track of, the following prompt evaluates their completeness:

> Given the following array of keys k and the result array that represents a binary tree. Check whether the tree is a complete binary search tree

> for the given keys. This means that all keys are present in the tree, and that the tree maintains a corrects binary tree structure. If so, answer with 1, if not, answer with 0. [...]
>
> Input: {input}
> Reasoning:
> Answer: Your answer here. Exactly one value of either 0 or 1 MUST be provided.

Afterwards, every solution that is considered to be complete, is added to a list of possible final solutions. The other solutions evaluated to be intermediate thoughts are then taken as the input for the next round of proposals, evaluation and completeness check.

At the very end of the algorithm, an evaluation is once again performed on the list of possible complete solutions to determine the most promising final solution. This final solution is then provided to the user as the output of the ToT algorithm.

3.4 Conducted Experiments

The ToT Algorithm is applied on the data set of 20 keys-frequencies pairs and the success rate is measured. For comparison, the data set is also tested on a 0-shot and a 3-shot prompt as well as the CoT approach.

For the 0-shot approach, the LLM is provided with the task and the keys-frequencies pair but with no further context or examples. The 3-shot approach additionally contains three whole examples of how to create the OBST for a given input, including the reasoning steps for better context. Finally, for the CoT approach, the prompt stimulates the model to build the final solution stepwise through intermediate thoughts. However, unlike the ToT approach, these thoughts are simply followed straightly without any further evaluation, backtracking functionality of similar.

4 Experimental Results and Evaluation

After running the experiments, the performance of every approach is being measured in order to compare how well ToT performs on the problem of creating OBSTs in comparison to 0-shot, 3-shot and CoT prompting. Additionally, an error analysis on the ToT approach is conducted to understand common errors and thus gaining a deeper understanding of the possibilities and limitations of this approach. Lastly, a cost analysis examines the average total costs that each prompting approach needs in order to generate a final solution.

4.1 Performance Analysis and Results

The analysis of the problem solving performance for creating OBSTs using ToT shows significantly different results compared to the results retrieved in previous papers on that matter [6, 14].

Fig. 4 shows the success rates of the different prompting approaches. As might be suspected, with 15% and 20% respectively, the success rates of the 0-shot and 3-shot prompts are moderately low. This shows that to date, the creation of OBSTs is a problem that is not trivially solvable by LLMs.

The CoT approach only achieves a success rate of 20% as well, providing no enhancement to the 3-shot prompt. However, these results so far are still closely related to the majority of results that could be observed in previous experiments that were based on different problems to solve. This shows that there are many problems that LLMs are not yet capable of solving easily, and the use of relatively simple prompt engineering approaches like 3-shot and CoT do not enhance these problem solving capabilities significantly.

In this context, the performance of ToT is very interesting: Unlike in previous experiments, ToT does not manage to increase the success rate significantly to over 50%, sometimes even 70 - 90%. Instead, with 25% of OBSTs created correctly, its success rate stays close to the 3-shot and CoT rates and only slightly increases by 5%. This strongly indicates that while there exists a set of problem types whose success rates can be increased impressively with ToT, this does not halt for any problem of arbitrary complexity.

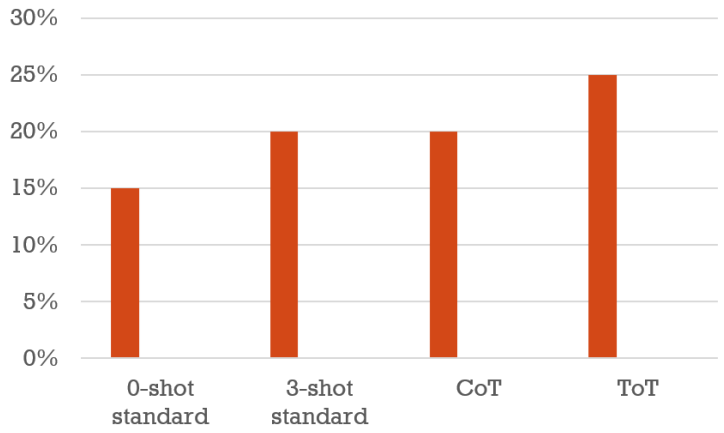


Fig. 4. Performance Analysis of ToT compared to 0-shot and 3-shot as well as CoT Approaches

4.2 Error Analysis

After the Performance Analysis has shown that ToT does not work exceptionally well on creating OBSTs, an error analysis can provide an understanding of the major difficulties this approach in cooperation with the underlying LLM faces.

The errors that occurred can be classified in four main error classes: syntactical and semantic errors, incomplete BSTs, and complete but not optimal BSTs.

Fig. 5 provides an overview of the frequencies of the different error classes. Syntactical errors are errors where the provided solution does not meet the required binary tree representation format at all. During the construction phase of the experiment, this error occurred significantly more frequently. However, with a more enforcing prompt design including the addition of specific examples on the binary tree representation in the most commonly used array-like format, as well as an upgrade of the available LLM version by OpenAI from gpt-3.5-turbo-0613 to gpt-3.5-turbo-1106 during the test design phase, the frequency of syntactical errors could be reduced to under 10%.

Semantic errors represent final solutions that are syntactically correct but do not meet the properties of a correct binary tree. 27% of all errors are semantic errors.

The most frequent form of errors with 55% are incomplete BSTs, i.e., trees that are syntactically correctly represented and also meet the properties of correct binary trees, but do not contain all keys that were given as input. Therefore, they cannot be OBSTs for the given input.

Lastly, if an output of the ToT algorithm is syntactically and semantically correct and contains all input key values as nodes in its tree, this solution might represent a complete and correct BST, but not an optimal BST. This case represents 7% of all errors.

This error analysis shows that over 80% of all errors occur in the correct representation of binary trees and their completeness. Therefore, the two most significant challenges in the creation of OBSTs with the ToT approach and LLMs can be identified: For an LLM that is trained on natural language, it is difficult and error-prone to correctly deal with and deeply understand the array representation of abstract data types like binary trees. Additionally, the understanding of how to incrementally build a binary tree in a number of different steps and therefore reach a complete binary tree with all input key values is a major difficulty for LLMs. Also, this difficulty cannot satisfactorily be addressed by the ToT approach.

4.3 Cost Analysis

The last factor that needs to be considered when evaluating the ToT approach as a whole is the cost of communicating with the LLM. While the exact costs for specific models vary, and even if the financial costs only play a subordinate role and e.g., time constraints are of main interest, the tokens sent to and received from the LLM are a good indicator for the resulting costs of a prompting method.

Therefore, the total number of tokens that were exchanged between the LLM and the requesting agent during the whole process of creating an OBST for a keys-frequencies pair was measured. Fig. 6 shows the number of exchanged tokens for the different prompting approaches. As this number also varies based on the number of input keys due to a different number of rounds in the algorithm, the measurement additionally splits the tokens according to the number of keys in

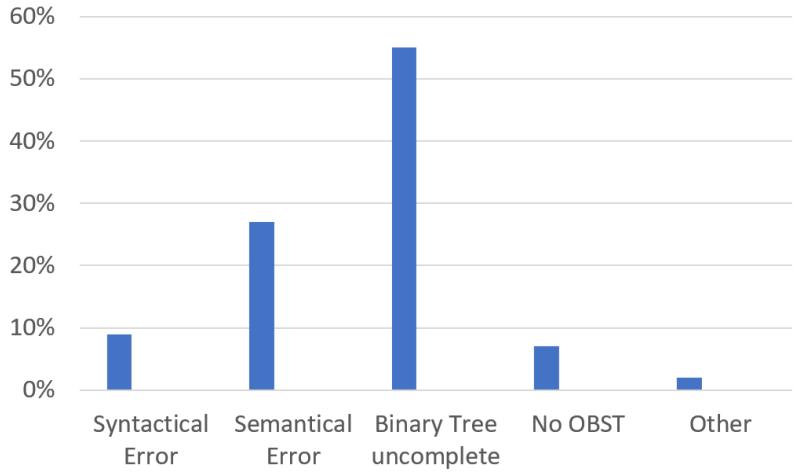


Fig. 5. Error Analysis of Experimental Results

the resulting OBST. It should be noted that the scale for the number of tokens is logarithmic.

The cost analysis clearly shows the significantly higher amount of tokens used in the ToT problem solving process in comparison to any of the other measured prompting approaches. While 0-shot, 3-shot and CoT need up to approx. 1000 tokens to provide a final solution, ToT already needs more than 10.000 tokens for OBSTs with only three keys, and more than 100.000 tokens for OBSTs with nine keys.

These higher costs can be explained through the structure of the ToT algorithm and its need to communicate with the LLM for thought proposals, evaluations and completeness checks. The costs could be reduced by various factors like a reduction of the maximal number of rounds in the algorithm, a reduced width of the spanned tree of thoughts or by shortening the prompts sent to the LLM. However, the fact that the costs will still be many times higher than the costs for 3-shot or CoT is inevitable and inherently lies in the characteristics of ToT.

5 Discussions and Future Work

In this paper, we further discussed the recently introduced ToT prompting approach. For this purpose, we conducted experiments on a more complex problem than the problems presented in previous papers. By applying ToT on the creation of OBSTs, we gained a deeper understanding of how the ToT algorithm works in detail, as well as its limitations in terms of complexity and costs.

For this particular problem, ToT only receives slightly better results than the CoT and 3-shot approaches. This suggests that while ToT can enhance the

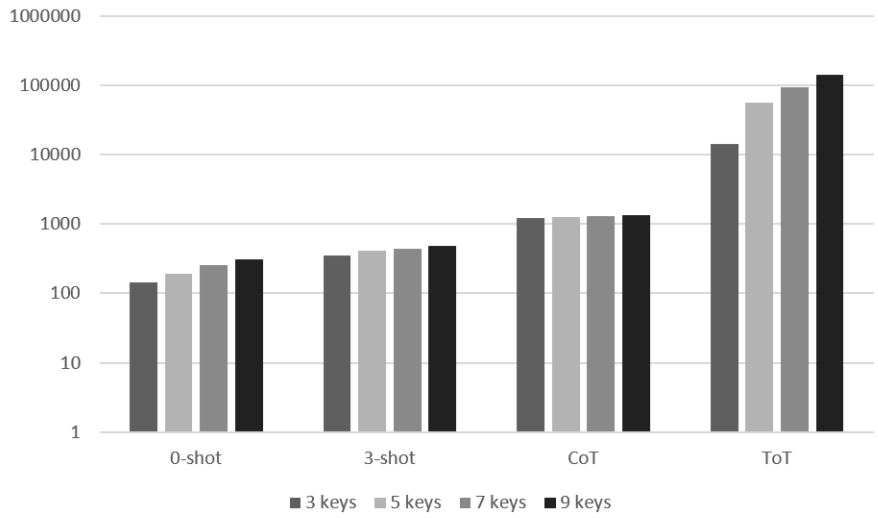


Fig. 6. Cost Analysis of ToT compared to 0-shot, 3-shot and CoT approaches

results for specific problems with certain characteristics, it is not capable of enhancing the results for any task of arbitrary complexity.

In future works, it might be interesting to further explore the set of characteristics a problem needs to fulfil in order to be applicable by ToT. Some of these characteristics that are needed to gain significant enhancements through ToT might be the availability of large data sets for this particular problem, a limited complexity of the task in general, a set number of steps to the final solution and a good problem representation in natural language.

Additionally, the significance of prompt engineering must be further explored for two reasons: Firstly, the impact of prompt qualities on the final results can be quantified. This will create a better understanding of whether locally optimal prompts can significantly enhance the results of ToT, or if rather the underlying algorithm is solely responsible for the final results of ToT. And secondly, prompt engineering has the potential to reduce the costs of ToT by optimizing and shortening user input prompts and enforcing outputs from the LLM that are as short as possible.

As a conclusion from the experiments conducted in this paper, and taking the results of previous publications into account, ToT is a promising prompting approach to enhance the problem solving capabilities of LLMs for certain tasks with specific characteristics. Nonetheless, a decision for each individual problem is necessary whether the increased problem solving capabilities justify the significantly higher costs.

References

1. Besta, M., et al.: Graph of Thoughts: Solving Elaborate Problems with Large Language Models (Nov 2023), http://arxiv.org/abs/2308.09687, arXiv:2308.09687
2. Hao, S., et al.: Reasoning with Language Model is Planning with World Model (Oct 2023), http://arxiv.org/abs/2305.14992, arXiv:2305.14992
3. Kim, G., Baldi, P., McAleer, S.: Language Models can Solve Computer Tasks (Nov 2023). https://doi.org/10.48550/arXiv.2303.17491, http://arxiv.org/abs/2303.17491, arXiv:2303.17491
4. Kojima, T., et al.: Large Language Models are Zero-Shot Reasoners (Jan 2023), http://arxiv.org/abs/2205.11916, arXiv:2205.11916
5. Lei, B., et al.: Boosting Logical Reasoning in Large Language Models through a New Framework: The Graph of Thought (Aug 2023). https://doi.org/10.48550/arXiv.2308.08614, http://arxiv.org/abs/2308.08614, arXiv:2308.08614
6. Long, J.: Large Language Model Guided Tree-of-Thought (May 2023), http://arxiv.org/abs/2305.08291, arXiv:2305.08291
7. Mialon, G., et al.: Augmented Language Models: a Survey (Feb 2023), http://arxiv.org/abs/2302.07842, arXiv:2302.07842
8. Nagaraj, S.V.: Optimal binary search trees (1997), https://community.wvu.edu/ krsubramani/courses/fa16/Aaoa/lecnotes/obst.pdf, chennai Institute of Technology
9. Ranaldi, L., Zanzotto, F.M.: Empowering Multi-step Reasoning across Languages via Tree-of-Thoughts (Nov 2023), http://arxiv.org/abs/2311.08097, arXiv:2311.08097
10. Shum, K., Diao, S., Zhang, T.: Automatic Prompt Augmentation and Selection with Chain-of-Thought from Labeled Data (Feb 2023). https://doi.org/10.48550/arXiv.2302.12822, http://arxiv.org/abs/2302.12822, arXiv:2302.12822
11. Wang, L., et al.: Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models (May 2023). https://doi.org/10.48550/arXiv.2305.04091, http://arxiv.org/abs/2305.04091, arXiv:2305.04091
12. Wang, X., et al.: Self-Consistency Improves Chain of Thought Reasoning in Language Models (Mar 2023). https://doi.org/10.48550/arXiv.2203.11171, http://arxiv.org/abs/2203.11171, arXiv:2203.11171
13. Wei, J., et al.: Chain-of-Thought Prompting Elicits Reasoning in Large Language Models (Jan 2023), http://arxiv.org/abs/2201.11903, arXiv:2201.11903
14. Yao, S., et al.: Tree of Thoughts: Deliberate Problem Solving with Large Language Models (Dec 2023), http://arxiv.org/abs/2305.10601, arXiv:2305.10601
15. Yao, Y., Li, Z., Zhao, H.: Beyond Chain-of-Thought, Effective Graph-of-Thought Reasoning in Large Language Models (May 2023). https://doi.org/10.48550/arXiv.2305.16582, http://arxiv.org/abs/2305.16582, arXiv:2305.16582
16. Zhao, W.X., et al.: A Survey of Large Language Models (Nov 2023), http://arxiv.org/abs/2303.18223, arXiv:2303.18223
17. Zhou, Y., et al.: Large Language Models Are Human-Level Prompt Engineers (Mar 2023). https://doi.org/10.48550/arXiv.2211.01910, http://arxiv.org/abs/2211.01910, arXiv:2211.01910