More Effectively Searching Trees of Thought for Increased Reasoning Ability in Large Language Models

Stanford CS224N Custom Project, Mentored by Tathagat Verma. Kamyar contributed to the project direction and topic, finetuning and LLM interfacing setup, and writing. Pranav contributed to experiment execution/analysis of results and writing. Sathya contributed to value function dataset creation/training and writing.

Kamyar Salahi
Department of Computer Science
Stanford University
`kamyar@stanford.edu`

Pranav Gurusankar
Department of Computer Science
Stanford University
`pranavg@stanford.edu`

Sathya Edamadaka
Department of Computer Science
Stanford University
`sath@stanford.edu`

Abstract

Large Language Models (LLMs) are capable of completing problem-solving tasks requiring different types of reasoning, but rely on token-level autogressive mechanisms for text generation and are limited in their ability to reason through complex, multi-step problems. The Tree of Thoughts framework extends Chain of Thought reasoning in LLMs to integrate a variety of possible options at each stage of reasoning. We introduce a new framework that extends the Tree of Thoughts approach by applying a separate value function that can more effectively evaluate reasoning paths and incorporating exploration from Monte Carlo Tree Search (MCTS). We find that vanilla tree of thoughts greatly outperforms a chain of thought based approach on the Game of 24, a task requiring non-trivial search through reasoning paths. However, upon incorporation of a separate finetuned value function, we find that the chain of thoughts based approach is actually able to match the solve rates of MCTS and BFS which use the LLM as a value function, but at a fraction (1/7) of the runtime. Consequently, we find that the efficacy and efficiency of a Tree of Thoughts reasoning is more contingent on the efficacy of the method to propose and evaluate paths rather than the search strategy through the paths themselves.

1 Introduction

‘Reasoning’ is an essential part of human cognition. Different kinds of reasoning are employed to work through different types of problems, and we can classify reasoning styles in a variety of ways - such as inductive vs. deductive vs. abductive, ’critical thinking’ or ’intuitive’, etc. Large language models (LLMs) like GPT (Brown et al., 2020) and PaLM (Chowdhery et al., 2022) are increasingly capable of completing tasks requiring different types of reasoning, and increased capacity to do so has mostly been as an emergent property of LLMs. However, until recently LLMs have largely relied on token-level autoregressive mechanisms for generating text (Yao et al., 2023), and this approach struggles through tasks that involve complex, multi-step reasoning through a problem Yao et al. (2023). LLMs as assistive agents in various contexts are becoming ubiquitous, a presence that will only increase and continue to be relied upon. It is therefore imperative that we find ways to better reasoning in LLMs, especially with ’complex’ problems that require step-by-step thinking and non-trivial planning.

In 2023, Yao et al drew inspiration from research by (Kahneman and Frederick, 2002; Sloman, 1996), to build a framework for LLM systems to engage in the "System 2" form of reasoning that humans use to

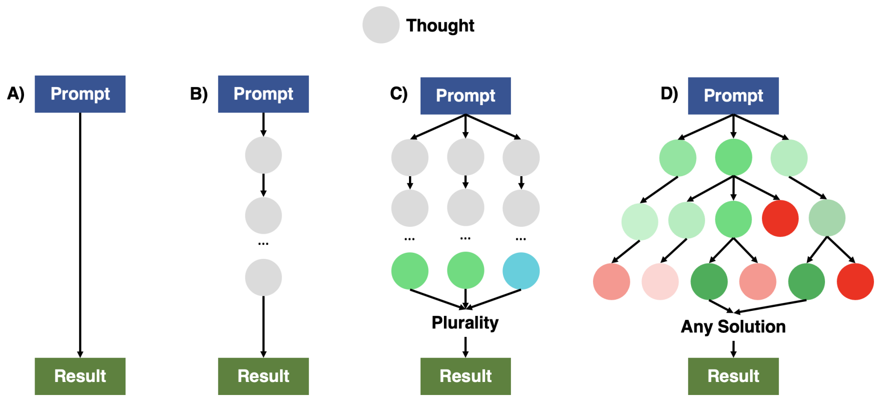

Figure 1: Different reasoning architectures via prompting, wherein a thought is represented as a circle and each thought is generated via further prompting. A) shows the default method of obtaining output. B) shows chain of thought prompting, in which a language model is prompted for a single step of the problem at a time until it finds a solution. C) shows the self-consistency chain of thought prompting, which takes a plurality vote in order to determine the output. D) shows the tree of thought (ToT) prompting framework, in which the model is prompted for several different potential next steps. The promise of each node is evaluated and scored, and a search algorithm is used to traverse the tree until a solution is found. This diagram is inspired by work from (Yao et al., 2023).

think through problems: a slow and step-by-step mode of thinking (Yao et al., 2023). On tasks which involve complex, multi-step reasoning and/or whose steps are not easily specified, the ToT approach demonstrates notable gains over existing reasoning strategies in language models. In the Game of 24 task, for example, utilizing GPT-4 with ToT achieved a success rate of 74% compared with the 4% rate achieved by GPT-4 with chain-of-thought prompting alone (Yao et al., 2023). While this result validates the ToT framework for complex reasoning in LLMs, an important limitation of this paper is that authors chose to explore relatively simple search algorithms for traversal of the reasoning tree after evaluating the likelihood of each node to lead to a solution. Specifically, they chose to explore the tree via either breadth-first search (BFS) or depth-first search (DFS). Although these are standard exploration algorithms, they don't guarantee a consistent, broad coverage of the tree of thoughts, instead either prioritizing a breadth of different options or pursuing single chains of thought deeply before backtracking. These approaches may also be relatively slow to find a solution, needing to explore a large number of nodes before finding a solution. We propose that a more advanced tree traversal algorithm, optimized for a larger degree of exploration without necessarily visiting as many nodes, may prove to be more performant.

A second important limitation is that in their ToT framework, Yao et al. utilize the LLM as the critic and evaluator for each thought candidate (to select top candidates for BFS). However, it is unclear if the LLM is able to reason well about the given task on its own, especially if such an explicit, prompting-based architecture like the ToT framework is needed to draw out effective reasoning skills. We propose that a finetuned LLM with specialized experience evaluating the likelihood of single reasoning steps would allow for more accurately scored trees of thought.

In this project, we look to extend the Tree of Thoughts framework in several ways. First, we incorporate Monte Carlo Tree Search (MCTS) as a search strategy for reasoning tree navigation, because the algorithm is well-poised to balance exploration and exploitation in large decision spaces (Kocsis and Szepesvári (2006)). This fulfils the design needs of our problem, and we call this approach MCToT: Monte Carlo search on Trees of Thought. Secondly, we come up with, and test, two different approaches to the value function used by the model: first, we evaluate our baselines and MCToT with an 'oracle' value function that references a database of all possible valid steps of the task at hand; secondly, we evaluate our baselines and MCToT with a value function we finetune specifically on the Game of 24 task. We also evaluate our models with trees of varying 'branching factors', or the number branches/reasoning paths at each thought junction. Overall, we find that on average we obtain the highest validation accuracies with MCTS and a tree branching factor of 5 and our finetuned value function. Our finetuned value function yields better results than both the default value function used by Yao et al. as well as the oracle. MCTS performs better than BFS for higher tree branching factors, while BFS performs marginally better for trees with lower branching factors. Overall, we find that the choice of evaluation method for reasoning paths (that is, our use of the value function) has a significantly greater impact on performance than the search algorithm used (BFS vs. MCTS).

2 Related Work

The poor reasoning capabilities of popular large language models have been commonly discussed in literature (Valmeekam et al., 2022). However, approaches to improving them have been extremely varied. Building off the default method of prompting an LLM for the entire solution at once (Figure 1A), Chain of Thought (CoT) prompting (Wei et al., 2023) forces the model to produce a single step at a time, more directly drawing upon its reasoning power (Figure 1B). CoT prompting underpins many reported methods for increased reasoning power, including domain-specific extensions to CoT prompting, like Program of Thoughts (Chen et al., 2022). Upon realizing that several chains of thought could be run in parallel and then combined in plurality (i.e. a majority vote for discretely valued solutions), self-consistent chains of thought (Wang et al., 2022) were proposed. Yao et al. observed that LLMs may benefit from more structured planning processes that not only explore diverse alternatives for current choices, but also evaluate the model's current status and look ahead for more global decision making, requirements not concurrently satisfied by any of Fig 1A-C (Yao et al., 2023). They proposed a tree of thoughts (ToT) framework for general problem solving with language models (Fig 1D). This ToT approach maintains several concurrent branches of thoughts, in which each thought is a coherent language sequence that serves as an intermediate step toward solving the original prompted problem. Least-to-most prompting involves directly getting the LLM to break the prompt into subproblems, solve them, and combine their solutions, but this proved to be roughly as performant as CoT prompting on a standard mathematical reasoning dataset (GSM8K) (Zhou et al., 2022). ToT is different from this in that the exact problem solving strategies and mathematical decomposition are left up to LLM and may change between steps, allowing for increased flexibility.

When deciding how our ToT model should make judgements on the promise of individual branches of thought, we looked to Zhu et al's CoRe, a zero-shot system that uses an LLM to generate proposals and several discriminator LLMs with different context lengths (single tokens, sentences, etc.) to decide which proposals have the most promise (Zhu et al., 2022). Drawing from Kahneman's work, they argued that this approach closely models the human brain's use of both System 1 and System 2 reasoning structures, particularly because they prompted LLMs for detailed reasonings along with the proposals themselves (Kahneman and Frederick, 2002). The idea to use multiple LLMs for different parts of the reasoning pipeline was extended by other works, finding improvement in reasoning by directly simulating planning frameworks with Monte Carlo tree searching to decide on potential actions. In particular, Hao et al proposed RAP, which uses LLMs as both world models and reasoning agents to generate a larger context about the task at hand and deliberately plan problem solving strategies via MCTS (Hao et al., 2023). Like CoRE, the state is communicated as the LLM gives a detailed explanation for its current global context. Similar planning approaches have found success in other tasks that heavily depend on reasoning, like code generation. In particular, Planning-Guiding Transformer Decoding was a method to allow transformers to use planners to better propose candidate programs and test them on provided test cases (Zhang et al., 2023). These methods demand significantly more resources and time than ToT approaches to not only run each LLM in parallel, but also train each of the planning framework components. However, MCTS was consistently reported as a highly performant tree exploration strategy that balances exploration and exploitation when deciding between potential steps, showing its promise in application to exploring a tree of thoughts.

An additional layer to proposing solutions to reasoning tasks is refinement: instead of settling for a single output to represent each branch of thought, Madaan et al. proposed Self-Refine, a method to iteratively improve an output based on self-feedback (Madaan et al., 2023). REFINER builds off of this idea, using self-refinement to improve intermediate representations of math problems (like trying to infer a formula from an initial question) (Paul et al., 2023). Consistent to several of this planning framework and self-refinement methods is using MCTS or another search algorithm to decide on potential branches of thought to explore (or, equivalently, potential actions to take). However, Chen et al's study on discerning when tree searching was actually useful for task planning revealed that high, underlying efficiencies of *discriminators* were prerequisites for MCTS-like methods to work well (Chen et al., 2024). A fundamental aspect of discriminator reasoning architectures is the method through which LLMs attribute value to proposed thoughts. Liu et al noticed that by using an explicit value function, specialized in judging potential branches of thought, in combination with MCTS, generation performance can be significantly improved (Liu et al., 2023). Therefore, training a value function that is specialized in making decisions related to reasoning tasks has been reported to have especially high merit.

3 Approach

In this project, we create a new framework for LLM inference that extends the original Tree of Thoughts framework by Yao et al. (2023) (and by derivation, the well-documented Chain of Thoughts (CoT) framework by Wei et al. (2023)). To summarize, these approaches involve maintaining several, independent branches of thought, in which the model is explicitly instructed to perform staged reasoning through an 'input' problem prompt. At each stage of reasoning, the model samples from multiple possible reasoning paths or 'thoughts', which form the branches of the tree. Each thought is a coherent language sequence that serves as an intermediate step toward solving the original prompted problem. A graphical illustration of the ToT framework and how it compares to previous reasoning strategies like CoT is shown in Fig 1.

In our approach, we utilize the ToT reasoning paradigm from Yao et al. (2023) but with the Monte Carlo Tree Search (MCTS) algorithm to search through the space of reasoning paths in reaching a solution to the input problem. Originally proposed by Kocsis and Szepesvári (2006), MCTS is a heuristic search algorithm commonly used in decision-making and optimization problems, and is notable for its ability to effectively balance exploration and exploitation in large decision spaces. The algorithm works as follows. First, a *selection* process is taken, which follows a branch of the tree until a final solution is obtained. Then, in *expansion*, unless a solution was obtained on the first DFS exploration, the algorithm backtracks to the second to last step. In *simulation*, another child of this step is explored. In *backpropagation*, information about whether or not the visited nodes thus far lead to solutions is distributed throughout other nodes in the tree, and the likelihood of finding a solution given information on other incomplete branches is updated. A flaw of this approach is that it may greedily not explore a branch of thought deemed to be unlikely to lead to a solution (even if it actually does) if a vast majority of other moves have not yet been explored. In our approach, we use the Upper Confidence Bounds applied for Trees (UCT) policy from the original Kocsis and Szepesvari work, which guides the selection of nodes to explore in MCTS away from this purely greedy strategy. The use of MCTS in the context of ToT reasoning is a novel exploration. We implement the ToT framework and MCTS from scratch, but with ideological inspiration from Yao et al. (2023) and Kocsis and Szepesvári (2006).

We also selected and implemented several baselines against which to measure our model performance. First, the 'vanilla' baseline in which we test a default LLM with Chain of Thought prompting. Our second and third baselines involve the same LLM, but with a Tree of Thoughts framework using the Breadth First Search (BFS) and greedy exploration algorithms respectively. These are the same baselines used to provide context by Yao et al's state-of-the-art work. Specifically, the 'vanilla' baseline is used by Yao et al. (2023), the ToT + greedy baseline is also referenced in the paper, and ToT + BFS is the primary algorithm implemented by Yao et al. (2023).

In our study, we evaluate each of our baselines and ToT models on the "Game of 24" task as evaluated by (Yao et al., 2023). The Game of 24 a mathematical reasoning challenge in which 4 non-negative integers, each less than 14, are provided as input. The player must find a way to combine the 4 numbers with arithmetic operations ($+$, $-$, $/$, $*$, and parentheses) to form 24. As a simple example, one may take the input $1, 1, 3, 8$ and form $1 * 1 * 3 * 8$, but as a more complicated example, one may take $2, 3, 5, 12$ and find $12/(3 - 5/2)$. The Game of 24 was chosen for evaluation of model reasoning in both the work by Yao et al. as well as this paper as it is a complex, multi-step task that requires non-trivial search and planning. To fit this problem into the ToT framework, the model is prompted to provide a single step of the process at a time. For instance, in the simple example, the first output may be "$1 * 1 = 1$. numbers left: $(1, 3, 8)$". In each output, two of the numbers are combined and one fewer number is left. A model has correctly found a solution if the final number left is 24 and if all of the intermediary steps are mathematically valid. An intermediate step is defined as the numbers left at the end of each operation.

4 Experiments

4.1 Data

Our dataset for the Game of 24 task, which we refer to as the '24nums' dataset, was scraped from `4nums.com`'s solutions page. We obtained all recorded 1362 problems and each of their viable solutions. Using these solutions, we parsed all possible intermediary steps used to generate 24. We then stored all valid intermediary steps as positive examples of valid paths to 24. For example, if a solution for an input $1, 1, 11, 11$ summed $1 + 1 = 2$ in its solution, we would store $2, 11, 11$ as a valid intermediary step to 24. However, in order to finetune a value function to evaluate potentially intermediate states, we also needed to produce negative examples. Rather than using randomly generated negative examples that may not fall into the distribution of possible visited states, we generated a set of hard negative examples by replacing each correct operation with every possible invalid one and continuing parsing as if it were a valid solution. Therefore, every negative example is only a single operator off of a positive one, ideally being close to positive branches in the tree of thought proposals. For example, $1 * 1 = 1$ is an operation that can be taken, but the resulting state $1, 11, 11$ is not a valid intermediate step to 24 (there is no means of producing 24 with those numbers). In this way, we were able to produce a set of 16,229 positive examples (with repeats) across all problems. After eliminating repeated valid intermediate states (as 138 is a valid intermediate state from starting problems of 1138 and 1335), we produced a training set of 906 unique positive examples and 906 negative examples. We also used these positive examples to construct a ground truth oracle which compares a given intermediate step with the set of all possible valid intermediate steps to determine if the path is viable or non-viable. In our evaluation of tree-based reasoning, we evaluate our approaches on the easiest 100 problems in the 24nums dataset (we used the percentage of players on `4nums.com` who solved the problem as a proxy for problem difficulty).

4.2 Evaluation method

We evaluate all of our approaches by how many successful runs they had out of the number of sets of four numbers they were given, the same success rate metric used by Yao et al. (2023).

$$\textbf{Success Rate} = \frac{\textbf{\# of correct final answers}}{\textbf{\# of test examples}} \tag{1}$$

A run was considered to be successful if any of the branches taken by the algorithm hit the state of 24.

4.3 Experimental details

4.3.1 Guardrails

For our experiments, we leveraged Mixtral-8x7B (Jiang et al., 2024) from the Together AI API unlike Tree of Thoughts (Yao et al., 2023) and many other approaches that leveraged the more expensive and more powerful GPT-4 models (OpenAI et al., 2024). In our initial experimentation, we found that the Mixtral-8x7B model was unable to solve any of the 10 easiest problems in the dataset using any of the branches it had taken. Upon further inspection, we found that two issues were resulting in this failure.

1. The model was hallucinating numbers inside of its input. If an input was $1, 1, 11, 11$ it may try to perform the operation $2 + 1 = 3$ and return $3, 11, 11$ as the next possible state.
2. The model was unable to perform arithmetic, producing invalid outputs that could not be generated using the arithmetic operations it provided. This is a known limitation especially for less powerful LLMs. In particular, our model of choice Mixtral-8x7B (Jiang et al., 2024) is able to reach 74.4% on GSM8K (Cobbe et al., 2021) with 8-shot chain of thought. In contrast, GPT-4 is able to achieve 92% accuracy on the same dataset with 5 shot chain of thought. Without chain of thought reasoning, these models are known to perform much worse (Wei et al., 2023). Since it is not possible to have both efficient and accurate chain of thought proposal of arithmetic operations while also leveraging tree of thought, our project did not leverage chain of thought prompting for operation proposal.

In order to resolve the aforementioned hallucination issues, we recrafted some of the prompts used in the original Tree of Thought paper to be more explicit with the desired task. In particular, rather than simply giving some examples, we also provided a prompt that said it was to perform arithmetic operations preceding the in-context examples of arithmetic operation proposals: *"Propose basic arithmetic (+ - * /) on two of the input numbers. Only provide numbers and operations in the desired format as shown."* We found that this helped improve hallucination to some extent, but we still suffered from a 0 solve rate on the easiest problems in our dataset.

Upon inspection, we found that the model produced invalid proposals (for its intermediate 'thoughts') approximately 19% of the time even after prompt engineering - providing numbers that were not in the set of input numbers. Consequently, we crafted guardrails on model outputs to ensure that they were valid. In particular, for every proposal generated by our model, we verified that the numbers were valid in that they were within the set of numbers available in the input. If they were not, we would skip those proposed arithmetic operations and resample. If they were, we would leverage python's evaluation function to compute the resulting value of the arithmetic expression (ie. eval("1+2") would return 3 which we could use to compute the resulting numbers in the following state) and proceed along the reasoning path. Our guardrails successfully captured the invalid proposals that were found after prompt engineering and replaced them with resampled, valid proposals.

4.3.2 Fine-tuning the Value Function

For both fine-tuning and evaluating our value function, we used Modal Labs' server-less compute platform. The model we finetuned was a pretrained Mistral-7B-Instruct-v0.1 available on Huggingface. To increase computational efficiency and reduce costs, we leveraged 4 bit quantization to reduce our GPU memory usage on the original model by around 59%.

Furthermore, rather than finetuning all of the weights, we utilized a low-rank adaptation module (Hu et al., 2022) with a rank of 64, an alpha of 16, and dropout probability of 0.1. We finetuned our model on an A10G with a learning rate of 1e-4 for 5 epochs on our dataset of 906 positive examples and 906 negative examples. Finetuning our model took 0.332 A10G GPU-hours. For our loss function, we leverage the same loss function used to train the reward model in reinforcement learning from human feedback (Ouyang et al., 2022). Concretely, our loss function is as follows:

$$loss(\theta) = -E_{(y_w, y_l)\sim D}[\log(\sigma(r_\theta(y_w) - r_\theta(y_l)))]$$

Here, y_w, y_l refer to sampled positive and negative example states respectively.

4.3.3 Evaluating Tree Search Approaches

We performed experiments evaluating model performance on the Game of 24 across our three value functions:

1. Using Mixtral-8x7B to evaluate how promising a path is with discrete values sure, likely and impossible. [the 'default' value function]
2. Using our finetuned Mistral-7B to evaluate how promising a path is with continuous floating point values between 0 and 100.
3. Using an 'oracle' value function to determine how promising a path is by assigning viable intermediate steps a value of 100 and inviable intermediate steps a value of 0.

As a reminder note, the 'oracle' value function references the database of all possible valid steps of the Game of 24 in informing the reasoning path chosen. When evaluating ToT across the three value functions, we also experimented with two different tree search algorithms (BFS and MCTS) and with different tree branching factors (branching factors of 1, 3, and 5 for BFS and branching factors of 3 and 5 for MCTS) to evaluate the degree to which the search strategy and branching factor will influence the success rate of the model. We found that due to the hallucination issues referenced in Section 4.3.1, vanilla CoT was not able to successfully solve any problems. Instead, we can consider that BFS with a tree branching factor of 1 is analogous to Chain of Thought (CoT) with an additional (aforementioned) value function(s) to guide decoding of arithmetic operations and next steps. For our default value function, tree search runs across our 100 examples took approximately 1.5 - 3 hours each. Our finetuned value function was in contrast much more efficient with a runtime of 20-40 minutes each.

4.4 Results

The results of our experiments across different value functions 'default', 'finetuned' and 'oracle', tree search strategies, and branching factors 1, 3 and 5 are summarized in Table 1. With respect to the relationship between branching factor and success rate, we found that increasing the branching factor generally resulted in higher success rates when the search strategy and value function were fixed. The only exception to this trend is observed for the default value function and MCTS, where increasing branching factor from 3 to 5 results in a small, but highly marginal drop in success rate.

Value Function	Search Strategy	Success Rate
Default Mixtral-8x7B	BFS-1	18.18%
Default Mixtral-8x7B	BFS-3	41.41 %
Default Mixtral-8x7B	BFS-5	66.66 %
Default Mixtral-8x7B	MCTS-3	48.48 %
Default Mixtral-8x7B	MCTS-5	47.47 %
Finetuned Mistral 7-B	BFS-1	47.47 %
Finetuned Mistral 7-B	BFS-3	54.54 %
Finetuned Mistral 7-B	BFS-5	63.63 %
Finetuned Mistral 7-B	MCTS-3	50.50 %
Finetuned Mistral 7-B	MCTS-5	65.65 %
Oracle	BFS-1	32.32 %
Oracle	BFS-3	57.57 %
Oracle	BFS-5	61.61 %
Oracle	MCTS-3	51.51 %
Oracle	MCTS-5	59.59 %

Table 1: Approach Success Rate

Value Function	Runtime (seconds)
Default Mixtral-8x7B	3.51711
Finetuned Mistral-7B	0.52849
Oracle	0.00002

Table 2: Runtime of Different Value Functions

Our results with respect to the relationship between the value function and model performance are more nuanced. Specifically, we found that when using Mixtral-8x7B to evaluate its own proposals [that is, the default value function] the success rate gap between using a tree based search strategy vs a chain based search strategy (ie. branching factor of 1) was more than 2x.

However, we found that these performance gaps were largely eliminated when we leveraged an oracle to evaluate viable paths and finetuned a value function ourselves. In particular we found that for BFS-1 and MCTS-5, model performance increases in the order of default->oracle->finetuned value function. For BFS-3 and MCTS-3, the finetuned value function yields better results than the default, but underperforms compared to the oracle. For BFS-5, the default value function marginally outperforms the finetuned and the oracle value functions. These results are mixed overall in trend, but the magnitude of performance difference is relevant. For example, the magnitude of performance difference in the cases where the finetuned value function underperforms relative to the oracle is highly marginal and is of at most 3.03%. Similarly, when the finetuned value function underperforms relative to the default value function in BFS-5 the difference in success rate is also 3.03%. However, the success rate gain when moving from the default value function to the finetuned value function on BFS-1 was 29.29%.

Interestingly, we found that the choice of search algorithm - that is, between BFS and MCTS - did not have a significant, generalized impact on the model's success rate. For example, MCTS-3 and MCTS-5 generally showed better performance than BFS-1 and BFS-3, respectively, but poorer performance than BFS-5 across value functions. Moreover, we found that the application of exploration in MCTS, especially with a poor value function, surprisingly resulted in a higher variance in our solve rate and consequently an occasionally lower solve rate.

Finally, we also found notable results with respect to value function choice and runtime performance. Specifically, we compared the runtime of the model across the 3 different value functions in evaluating a single, identical proposal in the Game of 24, with results summarized in Table 2. From this, we found that our finetuned value function was 7x faster than the default baseline.

5 Analysis

The results in Tables 1 and 2 illustrate several important trends. Firstly, they show that the effect of value function choice on model success rate is nuanced, with a greater effect on success rate than the choice of search algorithm (BFS vs. MCTS), but with an effect that is less pronounced with higher branching factors.

This tells us that the way in which the model chooses to evaluate individual proposals at intermediate steps of reasoning has a more significant impact on its overall task success than the choice of algorithm to search the space of reasoning paths. This intuitively makes sense: the model's success at searching the space of reasoning paths and 'pruning' out sub-optimal paths is contingent on its ability to evaluate the individual paths themselves. The effect of value function choice on success rate is smaller with higher tree branching factors, and this could be because with more 'options' generated at each reasoning step, there is a greater likelihood that at least one is a correct intermediate thought so there is less reliance on evaluating individual thoughts precisely with a value function, especially with an effective tree exploration strategy. Overall, however, we do see that the value function plays a bigger role in model performance than search algorithm choice. The finetuned value function generally has promising gains over the default, and especially in practical scenarios where an 'oracle' may not be feasible, it shows effectiveness in augmenting LLM reasoning.

Finally, to understand the reasons for the model underperforming in general, we looked at specific examples of intermediate outputs generated by our model on unsuccessful inputs and compared those intermediate outputs with the desired arithmetic operations. We attributed these 'failures' to two separate but often intertwined issues that occur within these methods. Firstly, the model was unable to propose the arithmetic operation required to get to the next intermediate level. In this case, no amount of increasing branching factor would improve performance. In the case of 9, 9, 11, 12, the model was unable to consider any of the valid intermediate outputs, so the value functions in all cases would be useless. This is responsible for the lower-than-expected success rate of the oracle runs despite having ground-truth, step-by-step solutions. Secondly, the model was able to propose an arithmetic operation required to move it forward toward computing 24 but would subsequently eliminate that path due to poor efficacy of the value function. In this case, increasing the branching factor allowed for more paths within the tree to be considered which enabled successful routes to be taken with higher probability. This is responsible for the increase in the performance of the model across increased branching factors.

6 Conclusion

Our results demonstrate that the success of Tree of Thought based reasoning can largely be attributed to the ability of these models to reason about the paths that they should take rather than the application of tree search in itself. Contrary to our expectations, we found that MCTS did not significantly outperform BFS when being used to find solutions in trees of thought, and in fact often had a higher variance in solve rates due to the exploration incentive. Despite the generally increased solve rates with a higher branching factor, we note that a significant boost in performance can be achieved by just leveraging a good value function. We purport that due to the computational cost of finetuning and evaluating a complex model as a value function, it is sensible to keep the discriminator model responsible for evaluating reasoning paths separate from the generator model, in line with current paradigms. Our qualitative evaluations demonstrated that failure cases across all approaches boiled down to the failure of the model to consider viable paths and the ability of the model to reason about the paths it proposed at all.

A limitation of our study is that we were only able to evaluate model performance on one complex reasoning task, the Game of 24. Further work might involve experimenting with LLM reasoning on other tasks, such as crosswords and other tasks that do not involve numbers. Another further exploration of this work might involve evaluating model performance for LLMs other than Mixtral-8x7B, such as GPT-4.

Overall, our work offers a notable contribution to the literature, as we demonstrate a way to further augment the ability of LLMs to reason through complex, multi-step problems with a thorough evaluation of intermediate reasoning 'thoughts'. Our approach to using a finetuned value function on the task provides generally better performance, but is importantly *much* more time-efficient than using an LLM to discretely evaluate the feasibility of its own reasoning paths. More generally, our findings about the relative importance of value function choice vs. search algorithm choice yield insights about reasoning in LLMs in general: that *how* an LLM 'reflects' on its choices in solving a problem is integral to its reasoning success.

References

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin

Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2022. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks.

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.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. Palm: Scaling language modeling with pathways.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2024. Mixtral of experts.

Daniel Kahneman and Shane Frederick. 2002. *Representativeness Revisited: Attribute Substitution in Intuitive Judgment*. Cambridge University Press.

Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. volume 2006, pages 282–293.

Jiacheng Liu, Andrew Cohen, Ramakanth Pasunuru, Yejin Choi, Hannaneh Hajishirzi, and Asli Celikyilmaz. 2023. Don't throw away your value model! making ppo even better via value-guided monte-carlo tree search decoding.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve

Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. 2024. Gpt-4 technical report.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744. Curran Associates, Inc.

Debjit Paul, Mete Ismayilzada, Maxime Peyrard, Beatriz Borges, Antoine Bosselut, Robert West, and Boi Faltings. 2023. Refiner: Reasoning feedback on intermediate representations.

Steven A. Sloman. 1996. The empirical case for two systems of reasoning. *Psychological Bulletin*, 119(1):3–22.

Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2022. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models.

Shun Zhang, Zhenfang Chen, Yikang Shen, Mingyu Ding, Joshua B. Tenenbaum, and Chuang Gan. 2023. Planning with large language models for code generation.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. 2022. Least-to-most prompting enables complex reasoning in large language models.

Xinyu Zhu, Junjie Wang, Lin Zhang, Yuxiang Zhang, Ruyi Gan, Jiaxing Zhang, and Yujiu Yang. 2022. Solving math word problems via cooperative reasoning induced language models.