问题树：通过组合性改进结构化问题解决

Armel Zebaze, Benoît Sagot, Rachel Bawden
Inria, Paris, France
`{ firstname.lastname } @inria.fr`

Abstract

Large Language Models (LLMs) have demonstrated remarkable performance across multiple tasks through in-context learning. For complex reasoning tasks that require step-by-step thinking, Chain-of-Thought (CoT) prompting has given impressive results, especially when combined with self-consistency. Nonetheless, some tasks remain particularly difficult for LLMs to solve. Tree of Thoughts (ToT) and Graph of Thoughts (GoT) emerged as alternatives, dividing the complex problem into paths of subproblems. In this paper, we propose Tree of Problems (ToP), a simpler version of ToT, which we hypothesise can work better for complex tasks that can be divided into identical subtasks. Our empirical results show that our approach outperforms ToT and GoT, and in addition performs better than CoT on complex reasoning tasks. All code for this paper is publicly available here: https://github.com/ArmelRandy/tree-of-problems .

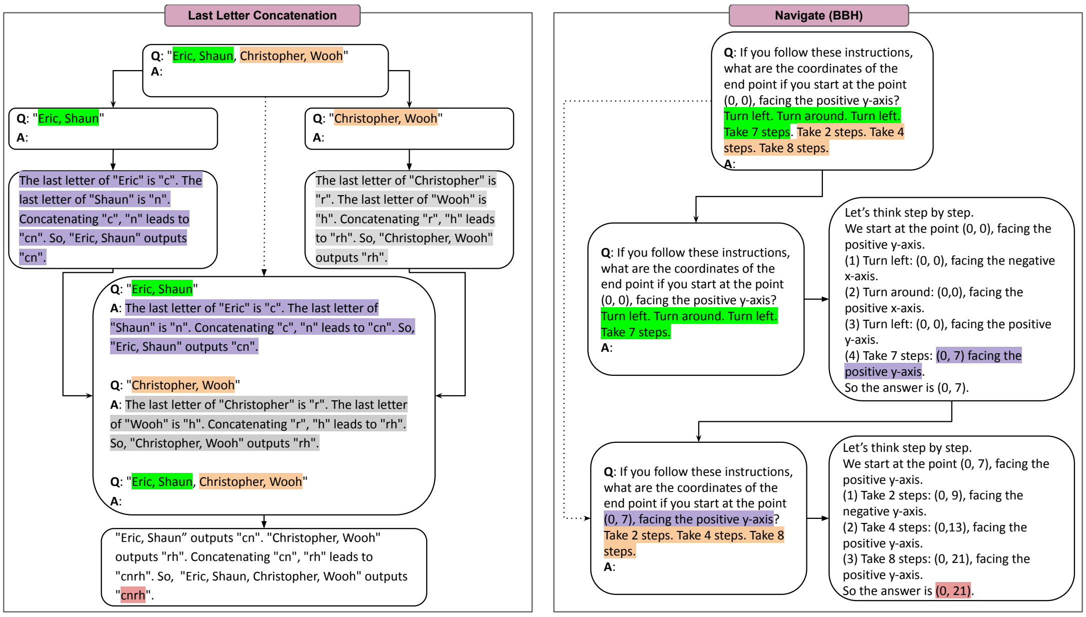
1 Introduction

In-Context Learning (ICL) (**?**) is the ability of Large Language Models (LLMs) to perform a task with the help of a few demonstrations within their context. It is widely used to evaluate LLMs on various tasks. These models, whose number of parameters and training corpus size has increased massively over recent years, keep pushing the state of the art on a wide range of natural language tasks (**???**) . However, they still struggle to perform complex tasks, notably those requiring multiple reasoning steps (**???**) . Recently, Chain-of-Thought (CoT) prompting (**??**) has greatly helped to enhance reasoning abilities of LLMs by helping them to mimic step-by-step reasoning. However, CoT implicitly requires the model to generalize beyond the cases seen in its prompt, which often leads to poor out-of-domain performance (**?**) . Applying CoT with self-consistency (**?**) drives the model to explore multiple reasoning paths and to choose the most consistent answer, usually yielding better performance, but helping only marginally with out-of-distribution generalization. Moreover, solving complex problems involves understanding their underlying structure; this can help to avoid lengthy CoTs that are prone to reasoning errors.

In this paper, we propose to tackle complex problem-solving and out-of-distribution generalization by dividing complex tasks into a series of simpler sub-tasks. We draw inspiration from techniques such as dynamic programming and divide and conquer in order to efficiently guide LLMs through complex problem solving. Such problems have previously been tackled using approaches adding structure to CoT, such as Tree of Thoughts (ToT) (**?**) and Graph of Thoughts (GoT) (**?**) , which consist in sampling diverse reasoning paths (where path states represent subproblems) and finding the optimal path. We argue that for a subset of complex reasoning problems, where an instance can be decomposed into multiple analogous subinstances, ToT and GoT are overly complex, and the tasks can be better solved by a simpler approach. This simpler approach, which we name Tree of Problems (ToP) consists in building a tree structure, where each node represents a problem instance similar to the main instance. The deepest instances, which correspond to atomic problems, are solved first with CoT prompting and the internal nodes are recursively solved by merging their children's solutions. Figure 1 illustrates our method on the tasks of Last Letter Concatenation and Navigate from the BIG-Bench Hard benchmark (**?**) .

We conduct a comprehensive evaluation on several LLMs, including GPT-3.5, on multiple hard tasks. We find that ToP improves LLMs' problem solving abilities on structured tasks outperforming CoT, ToT and GoT by a large margin.

www.xueshuxiangzi.com



Last Letter Concatenation

Q: "Eric, Shaun, Christopher, Wooh"
A:

Q: "Eric, Shaun"
A:

Q: "Christopher, Wooh"
A:

The last letter of "Eric" is "c". The last letter of "Shaun" is "n". Concatenating "c", "n" leads to "cn". So, "Eric, Shaun" outputs "cn".

The last letter of "Christopher" is "r". The last letter of "Wooh" is "h". Concatenating "r", "h" leads to "rh". So, "Christopher, Wooh" outputs "rh".

Q: "Eric, Shaun"
A: The last letter of "Eric" is "c". The last letter of "Shaun" is "n". Concatenating "c", "n" leads to "cn". So, "Eric, Shaun" outputs "cn".

Q: "Christopher, Wooh"
A: The last letter of "Christopher" is "r". The last letter of "Wooh" is "h". Concatenating "r", "h" leads to "rh". So, "Christopher, Wooh" outputs "rh".

Q: "Eric, Shaun, Christopher, Wooh"
A:

"Eric, Shaun" outputs "cn". "Christopher, Wooh" outputs "rh". Concatenating "cn", "rh" leads to "cnrh". So, "Eric, Shaun, Christopher, Wooh" outputs "cnrh".

Navigate (BBH)

Q: If you follow these instructions, what are the coordinates of the end point if you start at the point (0, 0), facing the positive y-axis? Turn left. Turn around. Turn left. Take 7 steps. Take 2 steps. Take 4 steps. Take 8 steps.
A:

Q: If you follow these instructions, what are the coordinates of the end point if you start at the point (0, 0), facing the positive y-axis? Turn left. Turn around. Turn left. Take 7 steps.
A:

Let’s think step by step.
We start at the point (0, 0), facing the positive y-axis.
(1) Turn left: (0, 0), facing the negative x-axis.
(2) Turn around: (0,0), facing the positive x-axis.
(3) Turn left: (0, 0), facing the positive y-axis.
(4) Take 7 steps: (0, 7) facing the positive y-axis.
So the answer is (0, 7).

Q: If you follow these instructions, what are the coordinates of the end point if you start at the point (0, 7), facing the positive y-axis? Take 2 steps. Take 4 steps. Take 8 steps.
A:

Let’s think step by step.
We start at the point (0, 7), facing the positive y-axis.
(1) Take 2 steps: (0, 9), facing the negative y-axis.
(2) Take 4 steps: (0,13), facing the positive y-axis.
(3) Take 8 steps: (0, 21), facing the positive y-axis.
So the answer is (0, 21).

Figure 1: Overview of the Tree of Problems (ToP) framework for two tasks. On the left (a canonical task consisting of independent subproblems organised in a tree structure), the task is to concatenate the last letters of a list of names, accomplished by breaking the list in two, finding their solutions, and recombining them. On the right (an extension of the canonical structure to handle sequential tasks), the task is to determine the final position of an object after a series of steps. We first find its position after half of the steps, and then determine the final position by tracing the object through the remaining steps. See Section 3 for a description of ToP.

2 Related Work

CoT prompting was proposed to enhance reasoning by incorporating step-by-step logic into few-shot prompt demonstrations (?) . It showed significant improvement over standard input-output (IO) prompting across various mathematical and symbolic reasoning benchmarks. Building on this, ? and ? inter alia demonstrated that zero-shot CoT could be achieved by using reasoning-inducing words at the end of the zero-shot prompt. Other works showed that wisely designing the CoT demonstrations could yield further improvements (??) . CoT Self-Consistency (CoT-SC; ?) improved on CoT by sampling diverse reasoning steps and selecting the most consistent answer after marginalizing over the reasoning paths. Our research also builds on the body of work addressing problem-solving through compositionality, which involves teaching LLMs to tackle complex problems by breaking them down into a series of subproblems and recursively solving them to derive the final answer, e.g. Least-to-Most (?) , decomposed (?) and successive (?) prompting. While these works align with our approach through their use of problem decomposition, we focus on breaking a main task into multiple similar subtasks, solvable using the same prompt. 此外，我们的方法使用了树结构，使问题解决过程更加灵活和覆盖更广。最密切相关的方法是思想树（ToT）(?) 和思想图（GoT）(?) 。ToT 基于采样多样化的推理路径的思想，但重新定义了问题解决，将其视为思想空间中的搜索，其中状态代表部分解决方案。GoT 通过包含思想聚合扩展了 ToT，这相当于我们的合并操作，并允许精炼 (?) 。虽然在他们的方法中“思想”代表一个一般的推理步骤，我们则聚焦于通过子问题进行推理。我们不在思想树上进行搜索，也不对树节点进行评分或精炼（改善）。相反，问题树中的每个节点都直接与解决问题相关，其自下而上的重新组合产生最终解决方案。因此，ToP 是一种比 ToT 和 GoT 更简单且成本更低的替代方案。

3 Our method

Solving a complex problem often requires reasoning, partly explaining the success of CoT prompting for such problems. Reasoning involves understanding a problem’s structure and design. This aspect is frequently overlooked in CoT because incorporating it can be challenging. Our method addresses this by constructing a tree of simpler, closely related subproblems to solve a more complex problem. We hypothesize that the capability of an LLM to solve simple instances can be ex-

www.xueshuxiangzi.com

tended to more complex ones. The ability of an LLM to solve a complex instance therefore lies in how accurately it can solve simpler ones and then combine their answers. The main class of problems we aim to tackle are complex problems that are divisible into independent subproblems resembling the initial one (we refer to these as canonical tasks). However, we also experiment with relaxing the independency constraint in order to tackle sequential tasks, which require finding the final state of a system after a series of independent processing steps (See the right of Figure 1). Our method relies on the following components:

工作流程可以描述如下：分解器构建问题树，求解器解决树叶子节点的子问题，合并器通过自底向上的方式递归地合并子节点的解决方案来得出每个节点的解决方案。推理调用的总次数（不包括问题分解的成本）等于树结构中的节点数。

In addition to canonical tasks with a classic tree structure (see the left of Figure 1), ToP can also be used for sequential tasks, where a given subproblem needs the result of a previous subproblem as an input (see the right of Figure 1). Our standard ToP paradigm described above can be used to solve such problems by setting the breadth to 1. This has the effect that the problem is decomposed into a sequence of n subproblems organised as hierarchy of depth n . When solving the $(k+1)$ -th subproblem, the solver will have access to its child subproblem's result, i.e. the result of subproblem k , thereby accounting for the sequentiality of the decomposition. The LLM is no longer required to merge subproblems' solutions; it is directly fed with a new problem formulation automatically computed using the corresponding child's solution. The final solution is obtained by solving the last subproblem, and so the main problem instance (root node) does not influence the inference cost.

For both tasks, all problems at the same level of the tree are solved in parallel to promote efficiency. We further detail the method with more examples in Appendix A .

4 Experiments

We first compare ToP to ToT and GoT to test our hypothesis that our simpler approach is more adapted to canonical tasks. We do this using the GoT tasks proposed by **?** . We then show that ToP is more effective in comparison to IO (direct input-output) and CoT prompting across a wider ranger of canonical tasks, namely Last Letter Concatenation (**?**) and 5 BIG-Bench Hard (**?**) tasks fitting the description. Finally, we test ToP on sequential tasks.

4.1 数据集

GoT tasks. **?** compared GoT to ToT, IO, and CoT prompting on three tasks (each with 100 examples): (i) Sorting , which involves arranging a list of 32 numbers ranging from 0 to 9 (both inclusive) in order, (ii) Set Intersection , which involves finding the common elements between two sets, each containing 32 elements and (iii) Keyword Counting , which involves identifying countries mentioned in a text and counting how many times each country appears.

Symbolic Reasoning. We use two toy tasks introduced by **?** (each with 500 examples): (i) Last Letter Concatenation , where the LLM is tasked with recovering the concatenation of the last letters from a list of names and (ii) Coin Flip , which evaluates if the LLM can deduce the final state of a coin (heads or tails) after people either flip it or not. During evaluation, we consider various list lengths (4, 8 and 16) for the first task, and different numbers of people involved (4, 8 and 16) for the second.

BIG-Bench Hard (BBH). BBH consists of 23 BIG-Bench (**?**) tasks that have been shown to benefit from CoT (**?**) . We use 8 tasks: [1] Boolean Expressions , Hyperbaton , Multi-Step Arithmetic Two , Navigate , Object Counting , Tracking Shuffled Objects (3, 5, 7) , Web of Lies and Word Sorting .

4.2 语言模型和提示

We experiment with `gpt-3.5-turbo` and `gpt-3.5-turbo-instruct` . [2] For the |solve_prompts|, we use the CoT prompts [3] of **?** on BBH tasks, with minor changes. The CoT prompts for Symbolic Reasoning are inspired by those in (**?**) , which contain 8 examples of 2-letters or 2-flips and those for GoT tasks are the same as in **?** . We report some implementation details in Appendix C and Appendix D .

---

[1] See Appendix D.2 for more details.

[2] More results and analysis for LLaMA (different model versions and sizes) are provided in Appendices B.1 and **??** .

[3] 我们在附录 B.4 中报告了一些关于 IO 的结果。

www.xueshuxiangzi.com

4.3 Main results

GoT tasks. Table 1 compares our results on the GoT tasks with those obtained by rerunning the CoT, ToT and GoT approaches from (?) . More precisely, we use the highest accuracy achieved with ToT and GoT on each task with `gpt-3.5-turbo-0125` . For Sorting , we intuitively choose $b = 2$ as in merge sort and $d = 2$ for performance. We use the same b for Keyword Counting , with $d = 4$ to get simple atomic instances. In Set Intersection , we use $b = 4$ because each set is divided into two disjoint subsets, resulting in four pairs of subsets (one pair per subproblem). Such a large breadth was sufficient to produce simple atomic problems, so we used $d = 1$. ToP outperforms ToT and GoT by a large margin on sorting with an absolute improvement of 40 % over GoT. Similarly, ToP outperforms GoT by 19 % and 5 % respectively on Set Intersection and Keyword Counting .

GoT Tasks	`gpt-3.5-turbo`			
	CoT	ToT (best)	GoT (best)	ToP (ours)
Sorting	0.02	0.17	0.28	**0.68**
Set Intersection	0.07	0.25	0.46	**0.65**
Keyword Counting	0.00	0.00	0.26	**0.31**

Table 1: Results on 3 tasks from (?) . In all results tables, best results are highlighted in bold.

Last Letter Concatenation. We consider ToP (2, 1). Subproblems are obtained by dividing the main list into b = 2 lists of equal length.

Last Letter Concatenation	`gpt-3.5-turbo-instruct`		
	IO	CoT	ToP (ours)
Four	0.032	0.900	**0.990**
Eight	0.000	0.662	**0.854**
Sixteen	0.000	0.252	**0.444**

Table 2: Results on Symbolic Reasoning tasks.

与从最少提示到最多提示和带有自我一致性的链式思维进行比较。 Least-to-most (L2M) prompting has also been successfully applied to Last Letter Concatenation (?) . Given a list of L names, L2M requires $L-1$ inference calls, the first to concatenate the first 2 last letters and the $L-2$ other to add the remaining last letters one after the other. Following ? , we provide a fair comparison of L2M to ToP by adapting ToP's tree structure to require the same number of inference calls as L2M. This is done by using trees of breadth 2 and depth $log_2(L) - 1$ for lists of length L . We compare ToP to L2M as well as CoT self-consistency with L reasoning paths. The results (Table 3) show that for L = 4 or L = 8, ToP (breadth = 2, depth = 1) achieves comparable performance to L2M while requiring half as many inference calls. When the number of inference calls is matched between the two methods, ToP demonstrates superior performance in all scenarios. CoT-SC lags behind both L2M and ToP.

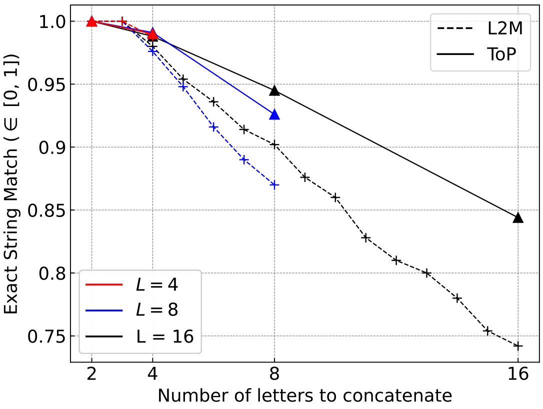


Figure 2: Per-level accuracy of Least to Most prompting and ToP (match) for Last Letter Concatenation .

Last Letter Concatenation	`gpt-3.5-turbo-instruct`			
	CoT-SC	L2M	ToP	ToP (match)
Four	0.908	0.988	0.990	**0.990**
Eight	0.574	0.870	0.854	**0.932**
Sixteen	0.116	0.742	0.444	**0.858**

Table 3: Comparison of ToP to L2M and CoT-SC for Last Letter Concatenation. ToP (match) refers to ToP with the same number of inference calls as L2M.

Moreover, since L2M is similar to ToP (1, $L-1$), we compare its accuracy to ToP (match) at each level of the tree. As illustrated in Figure 2 , both methods start with a perfect score that gradually decreases as they approach the task's resolution. ToP (match) consistently outperforms L2M at each step across all three settings.

4.4 Complementary results

We have successfully applied ToP to problems that can be divided into multiple independent instances. In this section, we report additional results for more such tasks and sequential tasks.

www.xueshuxiangzi.com

4.4.1 Canonical BBH tasks

BBH tasks such as Boolean Expressions , Hyperbaton , Multistep Arithmetic Two , Object Counting , and Word Sorting can be decomposed into multiple independent instances, whose solutions are later combined. They therefore correspond to canonical ToP tasks. We apply ToP (2, 1) to them and report results in Table 4 . ToP yields an absolute improvement over CoT of 21.2 % on Word Sorting and 9.8 % on Hyperbaton . However, it is slightly worse than CoT on Boolean Expressions , Multistep Arithmetic Two and Object Counting with an average deterioration of 3.6 % on the 3 tasks. We attribute this loss of accuracy to reasoning inconsistencies and we explore this in more detail in Appendix B.5 .

	`gpt-3.5-turbo-instruct`		
	IO	CoT	ToP
Boolean Expressions	0.908	**0.924**	0.896
Hyperbaton	0.528	0.804	**0.902**
Multistep Arithmetic Two	0.032	**0.780**	0.736
Object Counting	0.412	**0.928**	0.892
Word Sorting	**0.837**	0.619	0.831

Table 4: 在典型的 BBH 任务中的结果。

4.4.2 Sequential tasks

抛硬币是一个顺序任务的例子。使用 ToP (1, 2)，在叶节点的问题是找到硬币经过第一半人群之后的状态。最终的解决方案是通过确定硬币经过剩下的一半人群时状态如何变化来获得的。导航、跟踪打乱物体和谎言之网可以用类似的方式建模。如表 5 所示，ToP 在所有任务上都优于 CoT。ToP 在 4 人和 8 人的抛硬币任务上达到了近乎完美的准确度。此外，它在分布外设置中比 CoT 更加稳健，随着人数的增加性能下降较少。与 CoT 相比，它在谎言之网上有 5.2 % 的绝对提升，在跟踪打乱物体任务上平均有 5.9 % 的提升，在导航任务上有 2 % 的提升。

我们使用可组合性通过问题树（ToP）框架赋予大型语言模型（LLMs）解决复杂和结构化问题的能力。ToP 是简单化的任务树（ToT）框架，涉及将复杂任务分解为相同的子任务。我们的实验表明，LLMs 可以从 ToP 中受益，解决某些复杂问题的效果优于 ToT、GoT 和 L2M 方法，并且比使用 CoT 方法具有更好的泛化能力。

5

Acknowledgements This work was partly funded by the last two authors' chairs in the PRAIRIE institute funded by the French national agency ANR as part of the "Investissements d'avenir" programme under the reference ANR-19-P3IA-0001. The authors are grateful to the OPAL infrastructure from Université Côte d'Azur for providing resources and support. We thank Maciej Besta for answering our questions regarding Graph of Thoughts and Lydia Nishimwe for her helpful feedback.

	`gpt-3.5-turbo-instruct`		
	IO	CoT	ToP
Coin Flip			
Four	0.512	**0.998**	**0.998**
Eight	0.502	0.840	**0.998**
Sixteen	0.476	0.718	**0.756**
BIG-Bench Hard			
Navigate	0.204	0.864	**0.884**
Tracking Shuffled Objects (3)	0.004	**0.536**	0.524
Tracking Shuffled Objects (5)	0.004	0.324	**0.440**
Tracking Shuffled Objects (7)	0.000	0.044	**0.118**
Web of Lies	0.528	0.920	**0.972**

Table 5: Results on Coin Flip and sequential BBH tasks.

6

Limitations

Applicability of the Tree of Problems framework. Although ToP is a powerful prompting strategy that significantly widens the range of tasks that LLMs can handle accurately; it is limited to problems which have a structure (mathematical tasks, algorithmic tasks etc.) that can be decomposed into analogous subproblems. The founding hypothesis of ToP is the fact that LLMs can solve simple instances of a task and this ability can be efficiently translated to more complex instances.

Reasoning consistency of LLMs. LLMs can surprisingly fail to be robust to minor changes in a problem formulation. They can fail to solve a problem closely related to another one that they are capable to solve. We note this as a typical failure case of ToP in Appendix B.5 on Object Counting and Multistep Arithmetic Two .

www.xueshuxiangzi.com

References

A 澄清

A.1 Canonical Tasks

In Figure 1 we showed how to apply ToP $(2, 1)$ to an instance of Last Letter Concatenation. We illustrate how ToP $(2, 2)$ would look for concatenating the last letters of a list of 8 words in Figure 3 . The decomposition is done on two levels, the leaves being solved first and the merge operation being recursively applied from the bottom to the top.

A.2 Sequential tasks

Let us say that we have a system at state s_0 , and we want to find its state after going through m processing steps $(p_1, \ldots, p_m)$ in this order (i.e. a sequential task). Applying ToP $(1, k)$ is equivalent to grouping the above steps into k groups $G_1 = \left(p_1, \ldots, p_{\lceil \frac{m}{k} \rceil}\right), \ldots, G_k = \left(p_{m-\lfloor \frac{m}{k} \rfloor+1}, \ldots, p_m\right)$. We build a path graph from top to bottom, where the root is the main instance, and the leaf is the instance defined by s_0 and G_1 . Solving it yields a state s_1 to which we apply the steps G_2 and so on until we reach G_k . Tracking Shuffled Objects is an example of such a task. At the start, L people are assigned one object each. We are interested in recovering the assignment between people and objects after L swaps (transpositions). Figure 4 illustrates the application of ToP $(1, 3)$ to an instance with 3 swaps. We first decompose the main instance into 3 subinstances; here, each instance corresponds to one swap. After decomposition, only the first instance has the correct initial assignment (grey part). For the remaining instances, placeholders are used, which will later be replaced by the solutions to the problems they depend on.

A.3 Comparison with Least-to-Most Prompting

Least-to-Most prompting also handles Last Letter Concatenation as a sequential task. In this regards, it is similar to ToP $(1, L)$ on list with L words. As illustrated in Figure 5 , L2M uses all couples instance-solution preceding an instance to build the prompt to solve it whereas ToP only uses the couples directly connected to it in the tree hierarchy.

B 附加实验

B.1 Scaling behaviour

In this section, we study how ToP behaves as we vary the model scale. In Figure 6 , we plot the performance of both IO and CoT prompting and ToP as a function of model scale for LLaMA 2 models (?) and 3 BBH tasks. We use ToP (2, 1) for canonical tasks and ToP (1, 2) for sequential tasks. For all tasks, scaling up model size improved the performance of ToP beyond CoT prompting. LLaMA 2 70B achieves a 98 % accuracy on Object Counting , an absolute improvement of 18.8 % over CoT. ToP improves over random accuracy of IO and CoT on Web of Lies with LLaMA 2 7B, with an accuracy of 72.8 % .

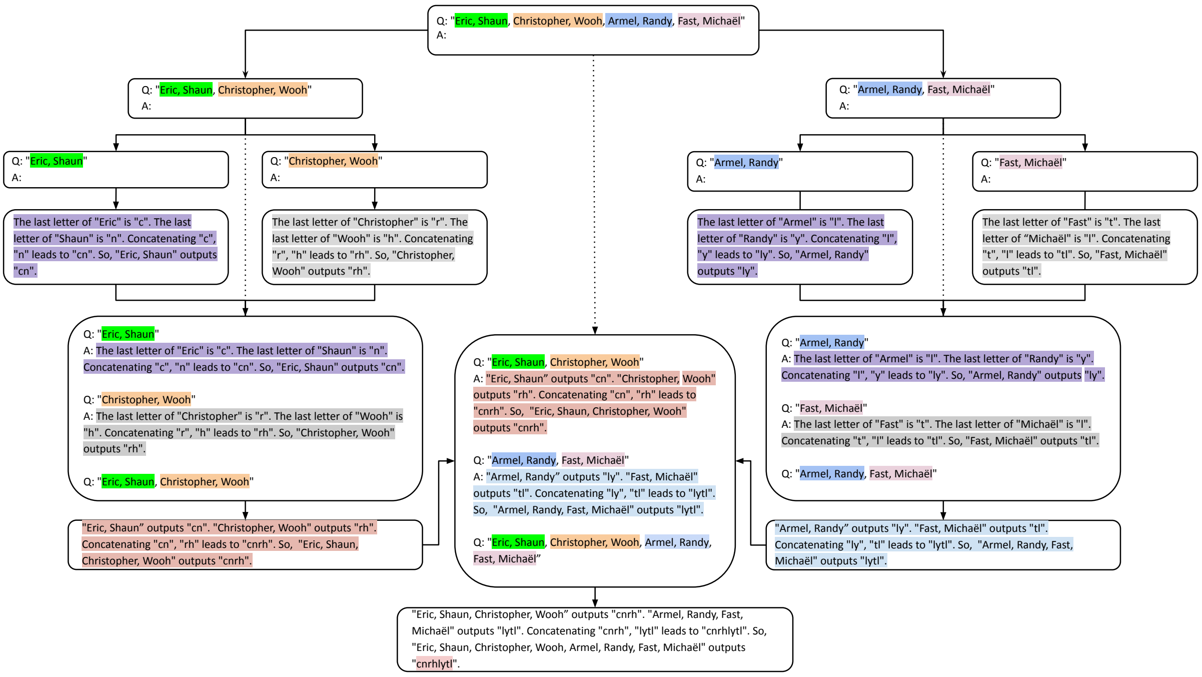
We report IO prompting, CoT prompting and ToP performance on 8 BBH tasks in Table 6 . ToP consistently yields an improvement of performance compared to IO and CoT prompting for most tasks and at all scales.

我们旨在全面理解 ToP 框架所带来的性能提升。我们从理论上导出了 ToP 性能的期望上界，然后研究了树结构对获得结果的影响。在本节中的实验中，除非另有说明，我们使用 LLaMA 3 8B (?) 。

B.2 Theoretical Analysis

Let us consider a task with n problems. Each problem is further divided into k subproblems, resulting in a total of nk subproblems. If we evaluate an LLM on these nk subproblems and obtain m incorrect answers, we can infer the number of incorrect answers likely to occur when evaluating the original n problems. Assuming that an incorrect answer to a subproblem implies an incorrect answer to its corresponding main problem, we can analyze the outcomes in two scenarios. In the worst case, each of the m incorrect subproblems is associated with a distinct main problem and thus there would be m main problems with incorrect answers. The best case is when the m incorrect subproblems are distributed such that each affected main problem has k or $m\%k$ incorrect subproblems. Consequently, the number of main problems with incorrect answers would be at most $\lceil \frac{m}{k} \rceil$. From this analysis, we can deduce that the accuracy at any level l of the problem hierarchy is constrained by the accuracy at level l - 1. Therefore, the accuracy for the overall task (the root of the hierarchy) is bounded by the accuracy observed at the most granular level (the leaves of the hierarchy). We

www.xueshuxiangzi.com



Q: "Eric, Shaun, Christopher, Wooh, Armel, Randy, Fast, Michaël"
A:

Q: "Eric, Shaun, Christopher, Wooh"
A:

Q: "Armel, Randy, Fast, Michaël"
A:

Q: "Eric, Shaun"
A:

Q: "Christopher, Wooh"
A:

Q: "Armel, Randy"
A:

Q: "Fast, Michaël"
A:

The last letter of "Eric" is "c". The last letter of "Shaun" is "n". Concatenating "c", "n" leads to "cn". So, "Eric, Shaun" outputs "cn".

The last letter of "Christopher" is "r". The last letter of "Wooh" is "h". Concatenating "r", "h" leads to "rh". So, "Christopher, Wooh" outputs "rh".

The last letter of "Armel" is "l". The last letter of "Randy" is "y". Concatenating "l", "y" leads to "ly". So, "Armel, Randy" outputs "ly".

The last letter of "Fast" is "t". The last letter of "Michaël" is "l". Concatenating "t", "l" leads to "tl". So, "Fast, Michaël" outputs "tl".

Q: "Eric, Shaun"
A: The last letter of "Eric" is "c". The last letter of "Shaun" is "n". Concatenating "c", "n" leads to "cn". So, "Eric, Shaun" outputs "cn".

Q: "Christopher, Wooh"
A: The last letter of "Christopher" is "r". The last letter of "Wooh" is "h". Concatenating "r", "h" leads to "rh". So, "Christopher, Wooh" outputs "rh".

Q: "Eric, Shaun, Christopher, Wooh"

"Eric, Shaun" outputs "cn". "Christopher, Wooh" outputs "rh". Concatenating "cn", "rh" leads to "cnrh". So, "Eric, Shaun, Christopher, Wooh" outputs "cnrh".

Q: "Eric, Shaun, Christopher, Wooh"
A: "Eric, Shaun" outputs "cn". "Christopher, Wooh" outputs "rh". Concatenating "cn", "rh" leads to "cnrh". So, "Eric, Shaun, Christopher, Wooh" outputs "cnrh".

Q: "Armel, Randy, Fast, Michaël"
A: "Armel, Randy" outputs "ly". "Fast, Michaël" outputs "tl". Concatenating "ly", "tl" leads to "lytl". So, "Armel, Randy, Fast, Michaël" outputs "lytl".

Q: "Eric, Shaun, Christopher, Wooh, Armel, Randy, Fast, Michaël"

Q: "Armel, Randy"
A: The last letter of "Armel" is "l". The last letter of "Randy" is "y". Concatenating "l", "y" leads to "ly". So, "Armel, Randy" outputs "ly".

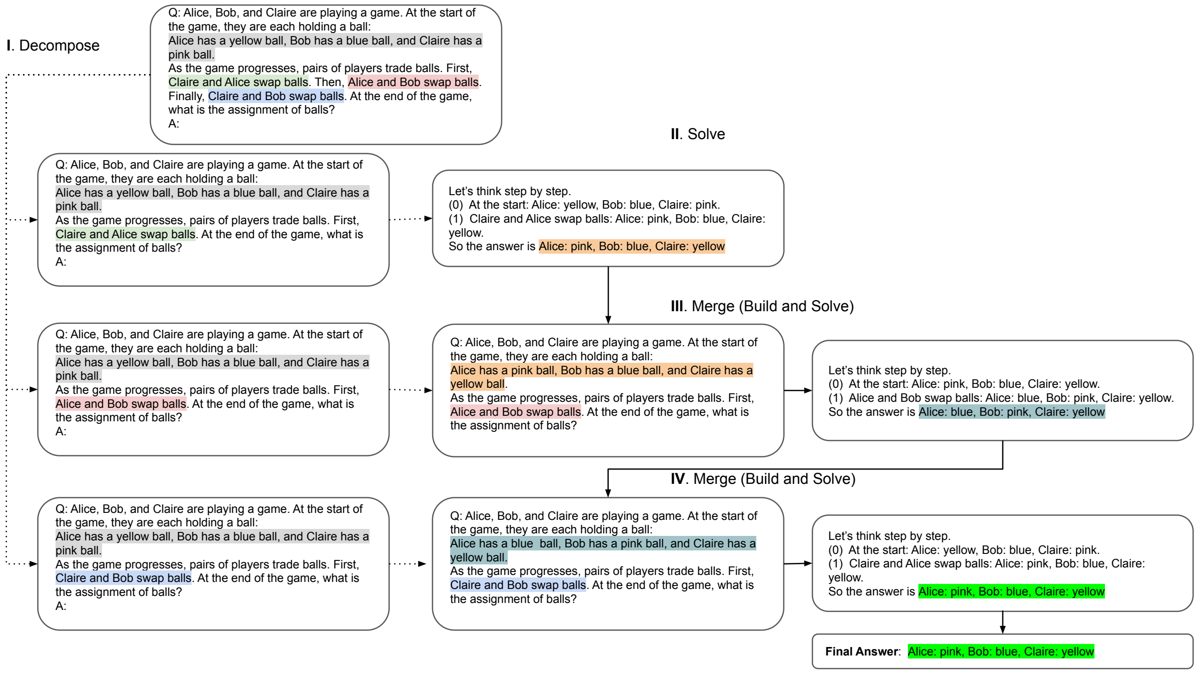
Q: "Fast, Michaël"
A: The last letter of "Fast" is "t". The last letter of "Michaël" is "l". Concatenating "t", "l" leads to "tl". So, "Fast, Michaël" outputs "tl".

Q: "Armel, Randy, Fast, Michaël"

"Armel, Randy" outputs "ly". "Fast, Michaël" outputs "tl". Concatenating "ly", "tl" leads to "lytl". So, "Armel, Randy, Fast, Michaël" outputs "lytl".

"Eric, Shaun, Christopher, Wooh" outputs "cnrh". "Armel, Randy, Fast, Michaël" outputs "lytl". Concatenating "cnrh", "lytl" leads to "cnrhlytl". So, "Eric, Shaun, Christopher, Wooh, Armel, Randy, Fast, Michaël" outputs "cnrhlytl".

Figure 3: Overview of ToP (2, 2) for Last Letter Concatenation. The list of words is divided into 2 sublists which are recursively divided into two sublists. The problems at the leaves, which consist into concatenating the last letters of 2-word lists are solved first. The solutions are then merged in a bottom-up way until the main instance is solved.



Q: Alice, Bob, and Claire are playing a game. At the start of the game, they are each holding a ball:
Alice has a yellow ball, Bob has a blue ball, and Claire has a pink ball.
As the game progresses, pairs of players trade balls. First, Claire and Alice swap balls. Then, Alice and Bob swap balls. Finally, Claire and Bob swap balls. At the end of the game, what is the assignment of balls?
A:

Q: Alice, Bob, and Claire are playing a game. At the start of the game, they are each holding a ball:
Alice has a yellow ball, Bob has a blue ball, and Claire has a pink ball.
As the game progresses, pairs of players trade balls. First, Claire and Alice swap balls. At the end of the game, what is the assignment of balls?
A:

Let's think step by step.
(0) At the start: Alice: yellow, Bob: blue, Claire: pink.
(1) Claire and Alice swap balls: Alice: pink, Bob: blue, Claire: yellow.
So the answer is Alice: pink, Bob: blue, Claire: yellow

Q: Alice, Bob, and Claire are playing a game. At the start of the game, they are each holding a ball:
Alice has a yellow ball, Bob has a blue ball, and Claire has a pink ball.
As the game progresses, pairs of players trade balls. First, Alice and Bob swap balls. At the end of the game, what is the assignment of balls?
A:

Q: Alice, Bob, and Claire are playing a game. At the start of the game, they are each holding a ball:
Alice has a pink ball, Bob has a blue ball, and Claire has a yellow ball.
As the game progresses, pairs of players trade balls. First, Alice and Bob swap balls. At the end of the game, what is the assignment of balls?

Let's think step by step.
(0) At the start: Alice: pink, Bob: blue, Claire: yellow.
(1) Alice and Bob swap balls: Alice: blue, Bob: pink, Claire: yellow.
So the answer is Alice: blue, Bob: pink, Claire: yellow

Q: Alice, Bob, and Claire are playing a game. At the start of the game, they are each holding a ball:
Alice has a yellow ball, Bob has a blue ball, and Claire has a pink ball.
As the game progresses, pairs of players trade balls. First, Claire and Bob swap balls. At the end of the game, what is the assignment of balls?
A:

Q: Alice, Bob, and Claire are playing a game. At the start of the game, they are each holding a ball:
Alice has a blue ball, Bob has a pink ball, and Claire has a yellow ball.
As the game progresses, pairs of players trade balls. First, Claire and Bob swap balls. At the end of the game, what is the assignment of balls?

Let's think step by step.
(0) At the start: Alice: yellow, Bob: blue, Claire: pink.
(1) Claire and Alice swap balls: Alice: pink, Bob: blue, Claire: yellow.
So the answer is Alice: pink, Bob: blue, Claire: yellow

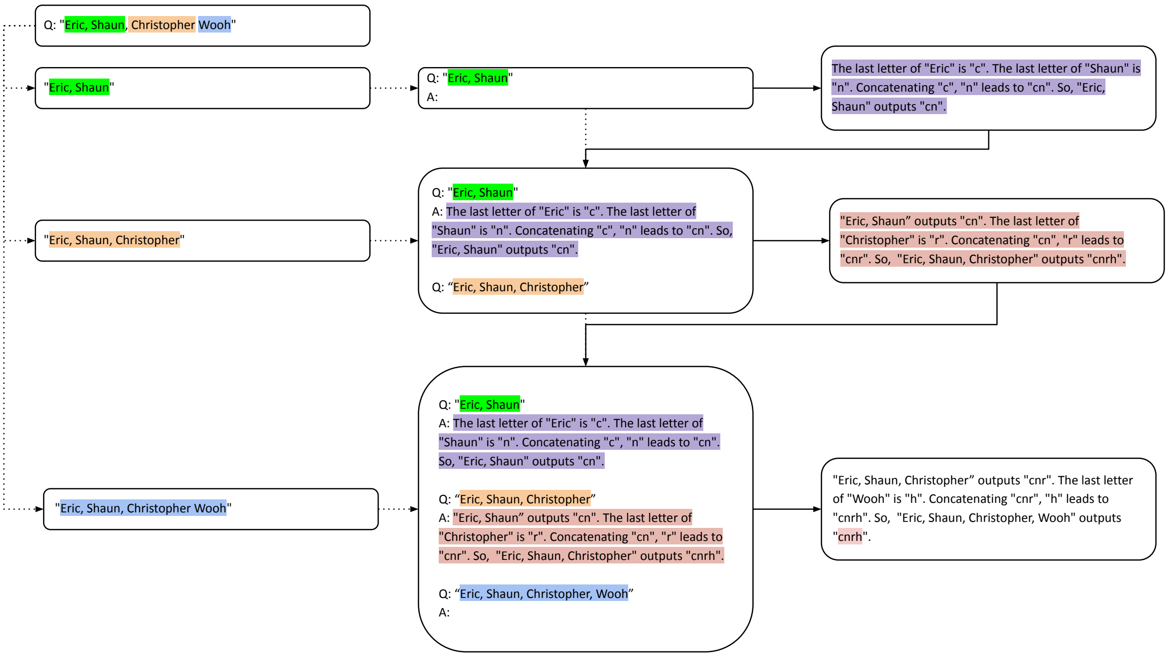
Final Answer: Alice: pink, Bob: blue, Claire: yellow

Figure 4: Overview of ToP $(1, 3)$ on an instance of Tracking Shuffled Objects (three objects) .

validate this analysis by comparing the accuracy at level 1 to the accuracy at level 0 (main problem) for some of the aforementioned BBH tasks. The results are summarized in Figure 7 . The Oracle Merger represents the accuracy that would be achieved if the merger process were flawless.

As expected, the accuracy at the leaves acts as an upper bound for ToP. Moreover, the Oracle Merger yields better performance than vanilla ToP. This suggests that there is a loss in accuracy when going from level k to level $k-1$, which can prevent ToP from achieving an even higher perfor-

www.xueshuxiangzi.com



Q: "Eric, Shaun, Christopher Wooh"

"Eric, Shaun"

Q: "Eric, Shaun"
A:

The last letter of "Eric" is "c". The last letter of "Shaun" is "n". Concatenating "c", "n" leads to "cn". So, "Eric, Shaun" outputs "cn".

"Eric, Shaun, Christopher"

Q: "Eric, Shaun"
A: The last letter of "Eric" is "c". The last letter of "Shaun" is "n". Concatenating "c", "n" leads to "cn". So, "Eric, Shaun" outputs "cn".

Q: “Eric, Shaun, Christopher”

"Eric, Shaun” outputs "cn". The last letter of "Christopher" is "r". Concatenating "cn", "r" leads to "cnr". So, "Eric, Shaun, Christopher" outputs "cnrh".

"Eric, Shaun, Christopher Wooh"

Q: "Eric, Shaun"
A: The last letter of "Eric" is "c". The last letter of "Shaun" is "n". Concatenating "c", "n" leads to "cn". So, "Eric, Shaun" outputs "cn".

Q: “Eric, Shaun, Christopher”
A: "Eric, Shaun” outputs "cn". The last letter of "Christopher" is "r". Concatenating "cn", "r" leads to "cnr". So, "Eric, Shaun, Christopher" outputs "cnrh".

Q: “Eric, Shaun, Christopher, Wooh”
A:

"Eric, Shaun, Christopher” outputs "cnr". The last letter of "Wooh" is "h". Concatenating "cnr", "h" leads to "cnrh". So, "Eric, Shaun, Christopher, Wooh" outputs "cnrh".

Figure 5: Overview of L2M prompting on Last Letter Concatenation with 4 words.

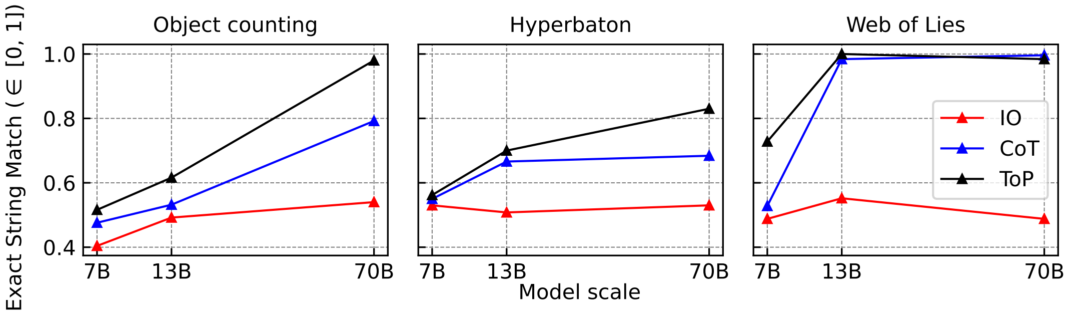


Figure 6: Scaling behavior of ToP compared to IO and CoT with the LLaMA 2 family on 3 BBH tasks.

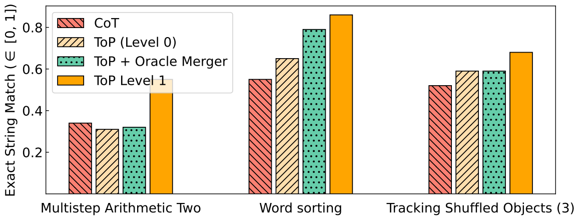


Figure 7: Comparison between CoT, ToP, ToP with an Oracle Merger and the leaves' accuracy on 3 BBH tasks.

mance. Interestingly, what happens with Multistep Arithmetic Two comes close to the worst case scenario that we depicted earlier. Despite the leaves' accuracy being 55 % , ToP + Oracle Merger fails to outperform CoT's 34 % accuracy, showing that the distribution of the correct leaves' instances inherently undermines ToP performance in this scenario.

B.3 Impact of the tree structure.

GoT Tasks. We analyze the impact of the tree structure on ToP's results. As shown previously, there may be a loss in accuracy during the merge operation. A deeper tree means more of these losses, but it also means easier subproblems. For the three GoT tasks, we analyze the impact of the tree's depth when the breadth is set to two with LLaMA 3 70B Instruct (**?**) .

As suggested by Figure 8 , deeper trees led to a higher accuracy for all three tasks. This is because we observed very few errors during the merge operation performed by the LLM. Going deeper, even with a near perfect merger can negatively affect performance as observed with the Set Intersection task, which has an accuracy of 47 %

www.xueshuxiangzi.com

BBH Tasks	**LLaMA 2 7B**			**LLaMA 2 13B**			**LLaMA 2 70B**		
	IO	CoT	ToP	IO	CoT	ToP	IO	CoT	ToP
Boolean Expressions	0.680	0.628	**0.672**	0.728	**0.768**	0.728	0.812	0.868	**0.924**
Hyperbaton	0.530	0.550	**0.562**	0.508	0.666	**0.700**	0.530	0.684	**0.830**
Multistep Arithmetic Two	0.008	0.004	**0.012**	0.012	0.024	**0.044**	0.016	0.196	**0.216**
Navigate	**0.272**	0.164	0.088	**0.340**	0.308	0.156	0.336	**0.400**	0.284
Object Counting	0.404	0.476	**0.516**	0.492	0.532	**0.616**	0.540	0.792	**0.98**
Tracking Shuffled Objects									
Three	**0.156**	**0.156**	0.136	0.076	**0.184**	0.132	0.056	**0.584**	0.568
Five	0.000	0.000	0.000	0.012	0.044	**0.048**	0.080	0.528	**0.664**
Seven	0.000	0.000	0.000	0.000	0.000	**0.004**	0.000	0.288	**0.592**
Web of Lies	0.488	0.528	**0.728**	0.552	0.984	**1.000**	0.488	**0.996**	0.984
Word Sorting	**0.418**	0.146	0.244	**0.538**	0.261	0.320	**0.788**	0.445	0.717

Table 6: LLaMA 2 系列在 BIG-Bench Hard (BBH) 上的少样本提示表现。

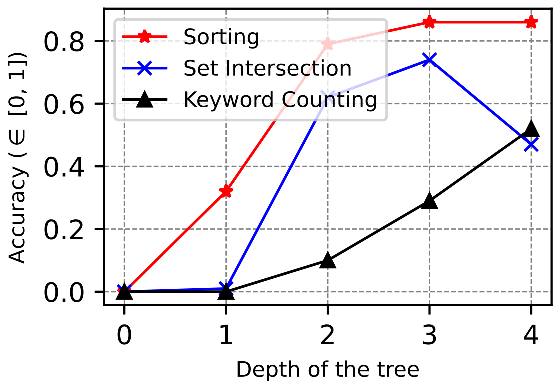


Figure 8: Impact of the tree structure (depth) on the accuracy on the 3 GoT Tasks with LLaMA 3 70B Instruct. Depth = 0 represents CoT prompting.

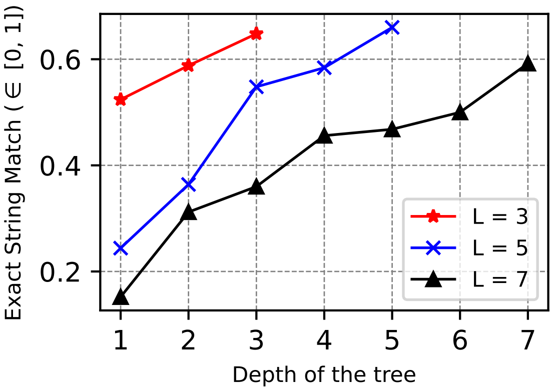


Figure 9: Impact of the tree structure (depth) on the accuracy of ToP on Tracking Shuffled Objects $(L \in \{3, 5, 7\})$. Depth = 0 and depth = 1 represent CoT prompting.

with $d = 4$ but 74 % with $d = 3$ and 62 % with $d = 2$. The small errors performed at the leaves being propagated during the repetitive merge operations impact the overall accuracy of ToP. In terms of breadth, applying ToP (4, 1) to Set Intersection yields the same accuracy of 62 % as ToP (2, 2). We observed ToP (4, 2) to have a 49 % accuracy, comparable to ToP (2, 4)'s 47 % .

BBH Tasks. 跟踪打乱的对象涉及在一系列 L 换位（成对交换）之后，恢复分配给 L 人的 L 对象的最终分配。将 ToP （1， d ）应用于这些任务意味着以类似于 Navigate 的方式使用 d 偶数子系列交换（见图 1 ）。我们研究了不同深度的影响，并在图 9 中报告了结果。

Across all settings, the task accuracy gradually increases with deeper trees and reaches its maximum when all the subproblems involve only one swap (depth = $L - 1$). The trade-off between the number of merge operations and the accuracy of simple instances is not at play here.

在多步算术问题二上，ToP (2, 1) 和 ToP (2, 2) 分别达到了 30.8 % 和 57.2 % 的准确率，而 CoT 的准确率为 34 %。同样地，在 Navigate 上，ToP (1, 2) 和 ToP (1, 3) 分别达到了 60 % 和 66.4 %，而 CoT 的准确率为 60.4 %。这表明树结构可以极大地影响 ToP 的质量。

B.4 Robustness to the solve prompt.

Throughout our experiments, we used CoT prompting to solve the most granular subproblems (the tree's leaves). In this section, we examine the impact of using IO prompting to solve the leaves. We conduct experiments on $WordSorting$, which did not benefit from CoT prompting as shown in Table 4 . Additionally, we include Tracking Shuffled Objects (3, 5) , Boolean

www.xueshuxiangzi.com

Expressions , Multistep Arithmetic Two , and Object Counting , where IO prompting produced much poorer results compared to CoT. The results are summarized in Table 7 .

BBH tasks	**LLaMA 3 8B**	
	IO	IO + ToP
Boolean Expressions	0.824	**0.876**
Multistep Arithmetic Two	0.008	**0.036**
Object Counting	0.492	**0.552**
Tracking Shuffled Objects		
Three	0.132	**0.196**
Five	0.004	**0.008**
Web of Lies	**0.528**	0.484
Word Sorting	0.647	**0.679**

Table 7: Comparison of IO prompting and IO + ToP on BBH tasks.

我们观察到，除谎言之网任务外，IO + ToP 在所有 BBH 任务上都持续优于单独的 IO 提示。这是因为在那个特定任务上，LLM 的表现是随机的（0.528），将其分解为更简单的实例并没有解决问题（leaves 的 EM = 0.496）。此外，在多步算术二问题中，将深度从 1 增加到 2，应用 IO + ToP 得到 22 % 的准确率。这些结果表明，ToP 对于低质量的 |solve_prompt| 具有鲁棒性，但在使用高质量的 |solve_prompt| 时能够取得最佳结果。

B.5 Analysis of failure cases

In Table 4 , we showed that ToP induces considerable performance gains on various complex BBH tasks with `gpt-3.5-turbo-instruct` . However, ToP specifically failed to improve performance on Multistep Arithmetic Two and Object Counting . An analysis of the model's outputs suggests 2 main causes:

1. ToP 的基本假设并不总是被验证。 ToP 的基础假设是 LLM 在任务的简单实例上比在复杂实例上表现得更好。当这一点未被验证时，ToP 的性能往往会落后于 CoT 提示。我们在对象计数上显著观察到这种行为。对于多个实例，LLM 能够在给定列表中计数相关项目，但在同一列表的子集中却无法做到这一点，如表格 8 所示。这种情况在多步骤算术中也经常发生。当有 73.6% 的 ToP 精度，即使有 Oracle 合并也不能得到提高。

MAIN INSTANCE
Q : I have three oranges, a pig, a frog, a cow, three bananas, a nectarine, and a snail. How many animals do I have?
Let's think step by step. We first identify the animals on the list and include their quantity in parentheses: - pig (1) - frog (1) - cow (1) - snail (1) Now, let's add the numbers in parentheses: 1 + 1 + 1 + 1 = 4. So the answer is 4.
SUBINSTANCE
Q : I have three bananas, a nectarine, and a snail. How many animals do I have?
Let's think step by step. We first identify the animals on the list and include their quantity in parentheses: - bananas (3) - 油桃 (1) - snail (1) Now, let's add the numbers in parentheses: 3 + 1 + 1 = 5. So the answer is 5.

Table 8: `gpt-3.5-turbo-instruct` 能够准确地计算给定列表中的动物数量，但在子列表中无法计算。

2. 大型语言模型无法将子问题的解决方案合并到父问题的解决方案中。 In Appendix B.2 , we showed that the use of an oracle merger leads to even greater improvements in the ToP framework. Additionally, a bad merger can bring ToP below CoT performance. Table 9 shows an example on the Object Counting task, where the LLM accurately solve both subproblems but fail to combine their answers. We believe that such issues can be significantly alleviated with a better |merge_prompt|.

C 实现细节

C.1 Language Models and Datasets

In Table 10 , we list the links to the relevant resources used to build this work.

C.2 Sampling scheme and evaluation metrics

For prompting strategies that require sampling, we use nucleus sampling with $T = 0.7$ and top-p $= 0.95$. Otherwise, we use greedy decoding ($T = 0.0$). In all experiments, we generate at most 2000 new tokens and use Exact String Match (EM) as the evaluation metric.

C.3 分解

All the decomposers are algorithmic. For sorting and set intersection, a list `L` is divided into

www.xueshuxiangzi.com

SUBINSTANCE 1
Q : I have a yam, a cauliflower, a bed, two cabbages, a garlic, and an oven. How many vegetables do I have?
Let's think step by step. We first identify the vegetables on the list and include their quantity in parentheses: - yam (1) - cauliflower (1) - cabbage (2) - garlic (1) Now, let's add the numbers in parentheses: 1 + 1 + 2 + 1 = 5. So the answer is 5.
SUBINSTANCE 2
Q : I have a carrot, a head of broccoli, a potato, a stalk of celery, a lettuce head, and a toaster. How many vegetables do I have?
Let's think step by step. We first identify the vegetables on the list and include their quantity in parentheses: - carrot (1) - broccoli (1) - potato (1) - celery (1) - lettuce (1) Now, let's add the numbers in parentheses: 1 + 1 + 1 + 1 + 1 = 5. So the answer is 5.
MAIN INSTANCE
Q : I have a yam, a cauliflower, a bed, two cabbages, a garlic, an oven, a carrot, a head of broccoli, a potato, a stalk of celery, a lettuce head, and a toaster. How many vegetables do I have?
Let's think step by step. The list of words can be written as the concatenation of A and B where - A = a yam, a cauliflower, a bed, two cabbages, and a garlic. - B = an oven , a carrot, a head of broccoli, a potato, a stalk of celery, and a lettuce head. According to the premises, A contains 5 vegetables. According to the premises, B contains 6 vegetables. Then, the total number of vegetables is A + B = 5 + 6 = 11 . So the answer is 11 .

Table 9: `gpt-3.5-turbo-instruct` 无法使用其上下文中的信息来解决所给的问题。

`L[0:len(L)//2]` and `L[len(L)//2:]` . For keyword sorting, we decompose the input text based on its sentences with the help of a sentence splitter module. [4]

C.4 How to choose the breadth and the depth

ToP depends on 2 parameters, the breadth and the depth of the tree structure. A quick analysis of the problem can lead to an informed guess about what a good breadth should be. This is typically the case of sorting problems when a breadth of 2 helps to mimic the merge sort algorithm. We mostly experimented with a breadth of 2 for canonical tasks and saw that it yielded very good results. When it comes to sequential problems, the breadth is 1 and the depth plays the role of the number of block of steps before reaching the final state (depth-wise decomposition). Using 2 blocks also gave good results, but deeper trees tend to always give better results for such problems.

D Prompts

D.1 GoT Tasks

We provide the links to all the prompts used to solve the GoT tasks in Table 11 .

D.2 BBH tasks

我们描述了对三个 BBH 任务的修改：Hyperbaton、Navigate 和 Tracking Shuffled Objects。我们将 Hyperbaton 的每个实例转变为相对于形容词顺序的两个独立的是/否问题实例，而不是选择两个句子中哪个具有正确的形容词顺序。表 12 展示了修改前后的一个实例（为了便于查看，我们只报告了一个独立的实例）。我们修改了 Navigate，以要求在一系列指令后找到最终位置，而不是询问这些指令是否导致起点。表 13 展示了修改前后的一个实例。在 Tracking Shuffled Objects 中，我们要求 LLM 在成对交换之后恢复最终的对象分配，如表 14 所示。

We provide the links to all the prompts used to solve the BBH tasks in Table 15 .

D.3 Symbolic Reasoning

We provide the links to all the prompts used to solve Coin Flip and Last Letter Concatenation in Table 16 .

---

[4] https://github.com/mediacloud/sentence-splitter

www.xueshuxiangzi.com

Datasets for Comparisons	
BIG-Bench Hard	https://huggingface.co/datasets/lukaemon/bbh
Models for Evaluations	
GPT-3.5-Turbo	gpt-3.5-turbo (gpt-3.5-turbo-0125)
GPT-3.5-Turbo-Instruct	gpt-3.5-turbo-instruct
LLaMA 3 8B	https://huggingface.co/meta-llama/Meta-Llama-3-8B
LLaMA 3 70B	https://huggingface.co/casperhansen/llama-3-70b-instruct-awq
LLaMA 2 7B	https://huggingface.co/meta-llama/Llama-2-7b-hf
LLaMA 2 13B	https://huggingface.co/meta-llama/Llama-2-13b-hf
LLaMA 2 70B	https://huggingface.co/TheBloke/Llama-2-70B-AWQ

Table 10: Links to datasets, benchmarks and language models.

CoT	
Sorting	https://github.com/ArmelRandy/tree-of-problems/blob/master/top/got/prompts/cot/sorting.txt
Set Intersection	https://github.com/ArmelRandy/tree-of-problems/blob/master/top/got/prompts/cot/set_intersection.txt
Keyword Counting	https://github.com/ArmelRandy/tree-of-problems/blob/master/top/got/prompts/cot/keyword_counting.txt
Merge	
Sorting	https://github.com/ArmelRandy/tree-of-problems/blob/master/top/got/prompts/merge/sorting.txt
Set Intersection (2)	https://github.com/ArmelRandy/tree-of-problems/blob/master/top/got/prompts/merge/set_intersection.txt
Set Intersection (4)	https://github.com/ArmelRandy/tree-of-problems/blob/master/top/got/prompts/merge/set_intersection_4.txt
Keyword Counting	https://github.com/ArmelRandy/tree-of-problems/blob/master/top/got/prompts/merge/keyword_counting.txt

Table 11: Links to solve and merge prompts of the GoT Tasks.

www.xueshuxiangzi.com

BEFORE
Q : Which sentence has the correct adjective order: Options: (A) rubber terrible ship (B) terrible rubber ship
IO Answer
(B)
CoT Answer
A : Let's think step by step. When there is more than one adjective before a noun, the adjectives need to respect the following order before a noun: "[1. opinion] [2. size] [3. age] [4. shape] [5. color] [6. origin] [7. material] [8. purpose] noun". Option (A): "rubber terrible ship". (1) rubber" falls into the material category. (2) "terrible" falls into the opinion category. Option (A) has the following adjective order: [7. material] [1. opinion] (or, in numeric terms, 7 1). Because $7 < 1$ is not correct, (A) does not have the correct ordering. Option (B): "terrible rubber ship". Option (B) has the following adjective order: [1. opinion] [7. material] (or, in numeric terms, 1 7). Because $1 < 7$ is correct, (B) has the correct ordering. So the answer is (B) .

AFTER
Q : Answer with Yes or No. Does the following sentence have the correct adjective order? rubber terrible ship
IO Answer
No
CoT Answer
A : Let's think step by step. When there is more than one adjective before a noun, the adjectives need to respect the following order before a noun: "[1. opinion] [2. size] [3. age] [4. shape] [5. color] [6. origin] [7. material] [8. purpose] noun". The sentence is "rubber terrible ship". (1) rubber" falls into the material category. (2) "terrible" falls into the opinion category. The sentence has the following adjective order: [7. material] [1. opinion] (or, in numeric terms, 7 1). Because $7 < 1$ is not correct, the sentence does not have the correct ordering. So the answer is No .

Table 12: 适用于倒装的修改。

www.xueshuxiangzi.com

BEFORE
Q : If you follow these instructions, do you return to the starting point? Turn left. Turn around. Turn left. Take 7 steps. Take 2 steps. Take 4 steps. Take 8 steps.
IO Answer
No
CoT Answer
A : Let’s think step by step. We start at the origin (0, 0), facing the positive y-axis. (1) Turn left: (0, 0), facing the negative x-axis. (2) Turn around: (0, 0), facing the positive x-axis. (3) Turn left: (0, 0), facing the positive y-axis. (4) Take 7 steps: (0, 7), facing the positive y-axis. (5) Take 2 steps: (0, 9), facing the positive y-axis. (6) Take 4 steps: (0, 13), facing the positive y-axis. (7) Take 8 steps: (0, 21), facing the positive y-axis. Since (0, 21) is not (0, 0), we are not where we started. So the answer is No .
AFTER
Q : If you follow these instructions, 如果你从点 (0, 0) 开始，面向正 y 轴，终点的坐标是什么？ Turn left. Turn around. Turn left. Take 7 steps. Take 2 steps. Take 4 steps. Take 8 steps.
IO Answer
(0, 21)
CoT Answer
A : Let’s think step by step. We start at the origin (0, 0), facing the positive y-axis. (1) Turn left: (0, 0), facing the negative x-axis. (2) Turn around: (0, 0), facing the positive x-axis. (3) Turn left: (0, 0), facing the positive y-axis. (4) Take 7 steps: (0, 7), facing the positive y-axis. (5) Take 2 steps: (0, 9), facing the positive y-axis. (6) Take 4 steps: (0, 13), facing the positive y-axis. (7) Take 8 steps: (0, 21), facing the positive y-axis. So the answer is (0, 21) .

Table 13: Modification applied to Navigate .

www.xueshuxiangzi.com

BEFORE

Q : Alice, Bob, and Claire are friends and avid readers who occasionally trade books. At the start of the semester, they each buy one new book: Alice gets Ulysses, Bob gets Frankenstein, and Claire gets Lolita. As the semester proceeds, they start trading around the new books. First, Claire and Bob swap books. Then, Bob and Alice swap books. Finally, Claire and Bob swap books.
At the end of the semester, Bob has
Options:
(A) Ulysses
(B) Frankenstein
(C) Lolita

IO Answer

(B)

CoT Answer

A: Let's think step by step.
(0) At the start: Alice: Ulysses, Bob: Frankenstein, Claire: Lolita.
(1) Claire and Bob swap books: Alice: Ulysses, Bob: Lolita, Claire: Frankenstein.
(2) Bob and Alice swap books: Alice: Lolita, Bob: Ulysses, Claire: Frankenstein.
(3) Claire and Bob swap books: Alice: Lolita, Bob: Frankenstein, Claire: Ulysses.
At the end of the semester, Bob has Frankenstein. So the answer is (B) .

AFTER

Q : Alice, Bob, and Claire are friends and avid readers who occasionally trade books. At the start of the semester, they each buy one new book: Alice gets Ulysses, Bob gets Frankenstein, and Claire gets Lolita. As the semester proceeds, they start trading around the new books. First, Claire and Bob swap books. Then, Bob and Alice swap books. Finally, Claire and Bob swap books.
At the end of the semester, what is the assignment of books?

IO Answer

Alice: Lolita, Bob: Frankenstein, Claire: Ulysses

CoT Answer

A : Let's think step by step.
(0) At the start: Alice: Ulysses, Bob: Frankenstein, Claire: Lolita.
(1) Claire and Bob swap books: Alice: Ulysses, Bob: Lolita, Claire: Frankenstein.
(2) Bob and Alice swap books: Alice: Lolita, Bob: Ulysses, Claire: Frankenstein.
(3) Claire and Bob swap books: Alice: Lolita, Bob: Frankenstein, Claire: Ulysses.
So the answer is Alice: Lolita, Bob: Frankenstein, Claire: Ulysses .

Table 14: 应用于跟踪混乱对象（三个对象）的修改。

	CoT
Boolean Expressions	`https://github.com/ArmelRandy/tree-of-problems/blob/master/top/bbh/prompts/cot/boolean_expressions.txt`
Hyperbaton	`https://github.com/ArmelRandy/tree-of-problems/blob/master/top/bbh/prompts/cot/hyperbaton.txt`
Multistep Arithmetic Two	`https://github.com/ArmelRandy/tree-of-problems/blob/master/top/bbh/prompts/cot/multistep_arithmetic_two.txt`
Navigate	`https://github.com/ArmelRandy/tree-of-problems/blob/master/top/bbh/prompts/cot/navigate.txt`
Object Counting	`https://github.com/ArmelRandy/tree-of-problems/blob/master/top/bbh/prompts/cot/object_counting.txt`
Tracking Shuffled Objects	`https://github.com/ArmelRandy/tree-of-problems/blob/master/top/bbh/prompts/cot/tracking_shuffled_objects.txt`
Web of Lies	`https://github.com/ArmelRandy/tree-of-problems/blob/master/top/bbh/prompts/cot/web_of_lies.txt`
Word Sorting	`https://github.com/ArmelRandy/tree-of-problems/blob/master/top/bbh/prompts/cot/word_sorting.txt`
	IO
Boolean Expressions	`https://github.com/ArmelRandy/tree-of-problems/blob/master/top/bbh/prompts/standard/boolean_expressions.txt`
Hyperbaton	`https://github.com/ArmelRandy/tree-of-problems/blob/master/top/bbh/prompts/standard/hyperbaton.txt`
Multistep Arithmetic Two	`https://github.com/ArmelRandy/tree-of-problems/blob/master/top/bbh/prompts/standard/multistep_arithmetic_two.txt`
Navigate	`https://github.com/ArmelRandy/tree-of-problems/blob/master/top/bbh/prompts/standard/navigate.txt`
Object Counting	`https://github.com/ArmelRandy/tree-of-problems/blob/master/top/bbh/prompts/standard/object_counting.txt`
Tracking Shuffled Objects	`https://github.com/ArmelRandy/tree-of-problems/blob/master/top/bbh/prompts/standard/tracking_shuffled_objects.txt`
Web of Lies	`https://github.com/ArmelRandy/tree-of-problems/blob/master/top/bbh/prompts/standard/web_of_lies.txt`
Word Sorting	`https://github.com/ArmelRandy/tree-of-problems/blob/master/top/bbh/prompts/standard/word_sorting.txt`
	Merge
Boolean Expressions	`https://github.com/ArmelRandy/tree-of-problems/blob/master/top/bbh/prompts/merge/boolean_expressions.txt`
Hyperbaton	`https://github.com/ArmelRandy/tree-of-problems/blob/master/top/bbh/prompts/merge/hyperbaton.txt`
Multistep Arithmetic Two	`https://github.com/ArmelRandy/tree-of-problems/blob/master/top/bbh/prompts/merge/multistep_arithmetic_two.txt`
Navigate	`https://github.com/ArmelRandy/tree-of-problems/blob/master/top/bbh/prompts/merge/navigate.txt`
Object Counting	`https://github.com/ArmelRandy/tree-of-problems/blob/master/top/bbh/prompts/merge/object_counting.txt`
Tracking Shuffled Objects	`https://github.com/ArmelRandy/tree-of-problems/blob/master/top/bbh/prompts/merge/tracking_shuffled_objects.txt`
Web of Lies	`https://github.com/ArmelRandy/tree-of-problems/blob/master/top/bbh/prompts/merge/web_of_lies.txt`
Word Sorting	`https://github.com/ArmelRandy/tree-of-problems/blob/master/top/bbh/prompts/merge/word_sorting.txt`

Table 15: 解决和合并 BBH 任务提示的链接。

www.xueshuxiangzi.com

CoT	
Coin	https://github.com/ArmelRandy/tree-of-problems/blob/master/top/coin/prompts/cot/cot8.txt
Concatenation	https://github.com/ArmelRandy/tree-of-problems/blob/master/top/concatenation/prompts/cot/cot8.txt
IO	
Coin	https://github.com/ArmelRandy/tree-of-problems/blob/master/top/coin/prompts/standard/standard8.txt
Concatenation	https://github.com/ArmelRandy/tree-of-problems/blob/master/top/concatenation/prompts/standard/standard8.txt
Merge	
Coin	https://github.com/ArmelRandy/tree-of-problems/blob/master/top/coin/prompts/merge/merge.txt
Concatenation	https://github.com/ArmelRandy/tree-of-problems/blob/master/top/concatenation/prompts/merge/merge.txt
L2M	
Coin	https://github.com/ArmelRandy/tree-of-problems/blob/master/top/coin/prompts/merge/l2m.txt
Concatenation	https://github.com/ArmelRandy/tree-of-problems/blob/master/top/concatenation/prompts/merge/l2m.txt

Table 16: Links to solve and merge prompts for Coin Flip and Last Letter Concatenation.

www.xueshuxiangzi.com