AN EXAMINATION ON THE EFFECTIVENESS OF DIVIDE-AND-CONQUER PROMPTING IN LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Foundation models, such as Large language Models (LLMs), have attracted significant amount of interest due to their large number of applications. However, when handling tasks involving repetitive sub-tasks and/or deceptive contents, such as arithmetic calculation and article-level fake news detection, simple instructional prompts suffer from inaccurate responses. Existing works show that more complicated prompting strategies, such as Chain-of-Thoughts and Least-to-Most, can unlock LLM's powerful capacity in diverse areas. Recent researches reveal that simple divide-and-conquer prompting strategy, i.e. simply dividing the input sequence to multiple sub-inputs, can also substantially improve LLM's performance in some specific tasks such as misinformation detection. In this paper, we aim at examining the utility of divide-and-conquer prompting strategy and answer on which kind of tasks this strategy gets advantages. Specifically, we provide a theoretic analysis to divide-and-conquer prompting strategy and help us identify the specific tasks where DaC prompting can bring performance boost with theoretic guarantee. We then present two cases (**large integer arithmetic and fact verification**) where experimental results aligns with our theoretic analysis.

1 INTRODUCTION

Large language models (LLM) based on the Transformer architecture have led to major breakthroughs in natural language processing and other related fields in artificial intelligence(Brown et al., 2020; Radford et al.; Touvron et al., 2023). State-of-the-art general-purpose language models have demonstrated remarkable advancements in various domains, including question answering, graph learning, reading comprehension, text generation, and machine translation (Chen et al., 2023b; Tan et al., 2023; Hendy et al., 2023; Mao et al., 2023; Zong & Krishnamachari, 2023). These developments paves the way towards general-purpose problem solvers (Bubeck et al., 2023).

However, as pointed out in (Wei et al., 2022), significant challenges arise when scale-up models are applied to tasks involved with long solution paths, such as those requiring mathematical or knowledge reasoning. A series theoretic works attribute this challenge to **Parallelism Tradeoff** (Merrill & Sabharwal, 2023a), a fundamental limitation of Transformers. Specifically, unlike Recurrent Neural Network whose computational depth is linear to the input sequence length (i.e., the depth is $O(n)$, where n is the input sequence length), Transformer does not contain any recurrent structure. Such design, while achieving superior parallelizability than RNN, makes Transformers suffer from limited expressive power. Merrill & Sabharwal proved that the expressive power of fixed-depth log-precision Transformer, which is very close to the most commonly applied Transformer architecture for LLMs, is bounded by constant-depth logspace-uniform threshold circuits. Thus, they fail to accurately tackle the tasks requiring long solution paths.

To address this challenge, carefully designed prompting strategies have been developed to tackle tasks that requires stronger expressive power (Feng et al., 2023). A series of works focus on prompting the LLM with instructions or context samples to output the intermediate steps that derive the final answer in an autoregressive manner, such as Chain-of-Thoughts (CoT) (Wei et al., 2022; Wang et al., 2022; Zhou et al., 2022; Chen et al., 2023a). Some works further apply programs to guide LLM to strictly follow designated reasoning steps (Yao et al., 2023). Theoretically, these prompting strate-

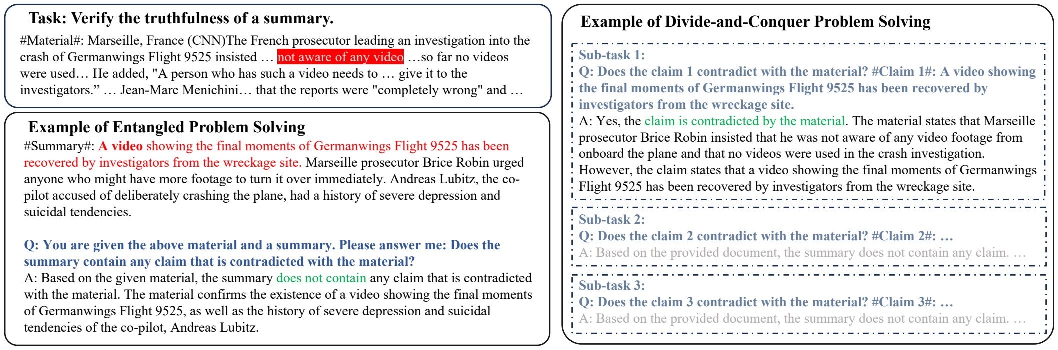


Figure 1: An illustrative example of hallucination detection with entangled problem solving (i.e., directly forward all inputs into the LLM) and divide-and-conquer problem solving (i.e., divide the problem inputs to parallel sub-tasks and tackle them parallelly). The sentence marked with red back font in the material is the evidence that contradict with the first claim in summary, which is marked with red font.

gies convert the role of Transformer from the complete problem solver to a sub-problem solver in a dynamic programming or tree searching algorithm (Merrill & Sabharwal, 2023b). In this way, these prompting strategies expand the expressive power of the LLMs and successfully improve the reasoning and searching of LLMs (Feng et al., 2023).

In contrast to such methods that apply instruction, context sample or program to decompose the whole reasoning process to multiple intermediate steps, in some tasks, researchers report that LLM's performance can also be boosted by simply **dividing the input sequences to multiple sub-inputs** and then merge the responses from LLMs on all sub-inputs, as shown in Fig. 1. For example, Cui et al. propose that in automated evaluation, LLM's performance can be further boosted by first dividing the input text to sentences and then evaluating them one by one. Intuitively, this paradigm benefits the tasks in a way similar to human brains, especially when the tasks are too hard or too complex. For example, when reviewing a long academic paper, some reviewers produce low-quality reviews (Garcia et al., 2021; Tennant & Ross-Hellauer, 2020; Cortes & Lawrence, 2021) containing hallucination-like **intermediate errors**, such as pointing out some 'missing baselines' that have already been sufficiently discussed by authors. To avoid such mistakes, experienced reviewers usually think slowly (Kahneman, 2011) to follow a **Divide-and-Conquer** paradigm to handle this task. Specifically, they decompose the paper review as examinations of multiple central opinions and then retrieve corpus to verify them respectively.

However, unlike Chain-of-Thoughts whose advance in expressive power is supported by theoretic analysis (Feng et al., 2023), the performance boost from Divide-and-Conquer paradigm is lack of rigorous theoretic support. As a result, we are not aware of the conditions under which the Divide-and-Conquer paradigm can acquire more accurate answers. To tackle this challenge, in this paper, we aim at understanding the utility of DaC prompting. More specifically, we attempt to answer the following two research questions:

1. **RQ1: Compared to straightforward instructional prompting, does DaC have theoretically guaranteed advantages similar as CoT and its variants?**
2. **RQ2: Compared CoT and its variants, what utility and limitations does DaC have?**

To answer these questions, we first provide a theoretic paradigm that can help us analyze how divide-and-conquer strategy expand the expressive power of fixed-depth log-precision Transformer on a given task. In this way, we provide a framework that can provide theoretic guarantee to DaC paradigm in various tasks. In this way, we present some conditions under which DaC have advantages compared to other prompting strategies. We then empirically evaluate DaC prompting and representative baselines on tasks that satisfy the proposed conditions and are challenging to existing prompting strategies even on state-of-the-art LLMs: Large Integer Multiplication, Hallucination Detection, Article-level Fact Verification (Cheng & Zhang, 2023; Li et al., 2023a; Wadden et al., 2020;

Hu et al., 2023; Wu et al., 2023). These tasks either require very long reasoning paths (e.g. large integer multiplication) or contain deceptive contents (e.g. hallucination detection and fact verification), making existing methods like Chain-of-Thought prompting prone to intermediate errors. Our experimental results show that the proposed method outperforms the baselines on all three tasks, which supports our theoretic analysis.

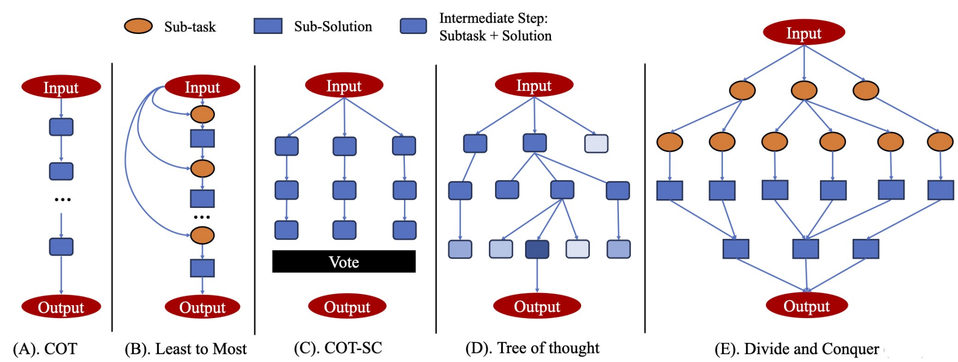


Figure 2: The comparison between DaC and the existing methods for prompting. The ellipse marks represent sub-tasks, the right-angled rectangles represent sub-task solutions, and the rounded rectangles represent intermediate steps that entangle sub-task and sub-solutions. The different shades in Tree of Thoughts (subfigure D) indicate the rates of different search directions. In CoT (Chain-of-Thoughts), CoT-SC and ToT, the Large Language Models must simultaneously generating and resolving sub-tasks. Least-to-Most (also Decomposed Prompting) disentangle sub-task generation and resolution. However, its sub-task resolution and resolution assembly process are intertwined as it sequentially attach new sub-tasks onto the previous resolution. Different from them, DaC totally disentangle the sub-task generation, sub-task resolution, and resolution assembly process.

2 Related Work

2.1 Expressive Power of Transformer

As discussed in previous works (Merrill & Sabharwal, 2023a; Feng et al., 2023), the expressive power of fixed-length log-precision transformers, which are widely applied in modern Pre-trained Large Language Models, is actually much more limited than people's expects. Merrill & Sabharwal give a theoretic proof that the expressive power of fixed-length log-precision transformers is upper-bounded with TC^0. Feng et al. further extend their analysis to explain that a lot of common problems exceed the expressive power of fixed-length log-precision transformers. Such results explains why the powerful LLM may make some ridiculous mistakes and how CoT improve the performance.

2.2 Prompting Strategies of LLM

In this sub-section, we introduce the existing prompting and discuss their limitations and drawbacks. Following the notations in (Yao et al., 2023), we denote the Large Language Models with parameter θ as p_θ and use lower case letters x, y, z to denote input sequence, result, and intermediate steps, respectively.

Input-Output (IO) Prompting is the standard prompting strategy that attach input x with instructions and/or few-shot in-context-learning examples to aqcuaire a prompt, denoted as $\mathsf{prompt}(x)$ (Yao et al., 2023). The LLM takes $\mathsf{prompt}(x)$ as input and predict result, i.e. $y \sim p_\theta(y|\mathsf{prompt}(x))$.

Chain-of-Thought (CoT) Prompting (Wei et al., 2022) aims at simulating human's thinking process that handles complicated task (e.g. combinational reasoning and mathematical calculation) in a step-by-step manner. More specifically, the LLM is guided to output a series of intermediate steps $z_1, z_2, ..., z_n$ (also known as *thoughts*) autoregressively, i.e. $z_i \sim p_\theta(z_i|\mathsf{prompt}(x), z_1, ..., z_{i-1})$. Then the LLM output the prediction of result y based on the *thoughts*, i.e. $y \sim p_\theta(z_i|\mathsf{prompt}(x), z_1, ..., z_n)$.

Exploration-of-Thought (EoT) Prompting and **Program-guided Prompting** are two variants of CoT. EoT includes a series of CoT's variants, such as Self-consistency with CoT (CoT-SC) prompting (Wang et al., 2022) and Tree-of-Thoughts (ToT) prompting (Yao et al., 2023), which aim at addressing the limitation of CoT in exploration. Their common central idea is to generate multiple chains of thought through sampling or proposing prompting and then ensemble them to acquire a final prediction. Program-guided Prompting aims at controlling the LLM's generation process with symbolic programs or pre-defined procedure (Zhu et al., 2022; Jung et al., 2022; Zhou et al., 2022; Khot et al., 2022; Creswell & Shanahan, 2022; Gao et al., 2023). Among them, the Least-to-Most (LtM) Prompting (Zhou et al., 2022) and Decomposed Prompting (Khot et al., 2022) are close to this work. They are the earliest attempts that explicitly prompt the LLM to decompose the task as a series of sub-tasks and sequentially tackle them. LtM prompt a LLM to iteratively raise sub-tasks and sequentially solve them to acquire the final resolution. Decomposed Prompting can regarded as a upgraded version of LtM. It introduces special notations into the prompt to represent program states and thus can call itself (i.e., recursion) or other modules (i.e., hierarchical decomposition), endowing it stronger expressive power. Such design increased the compositional generalization ability of LLMs in different areas, such as symbolic manipulation and multi-hop QA (Khot et al., 2022).

The aforementioned CoT and EoT families incorporate LLM with stronger expressive power than IO prompting. However, a critical issue of them is that, they could miss or ignore some important intermediate steps or contents (Liu et al., 2023). This problem is even worse when we are handling tasks involved with long input (e.g. long documents and large numbers). Typical examples include large number arithmetic calculation and fact verification in long documents. Compared to them, Least-to-Most prompting and Decomposed Prompting introduce explicit task decomposition to enumerate sub-tasks. However, their task decomposers are based on multi-round conversation or question-answering, which navigate the LLM through the deceptive content's flow sequentially, and propagate the hallucination/deception in the contexts (Dziri et al., 2024; Yang & Ettinger, 2023), leading to decreased performance.

3 Preliminary of Divide-and-Conquer Prompting

In this section, we summarize and formalize **Divide-and-Conquer prompting strategy**. Divide-and-Conquer prompting strategy consists of three distinct stages: task decomposition stage, sub-task resolution stage, solution merge stage. In task decomposition stage, the LLM is prompted to explicitly decompose the task as a series of parallel homogeneous sub-tasks with smaller problem sizes (e.g. divide a long paragraph to sentences). Such design avoids the multi-round conversation or question-answering in LtM and Decomposed Prompting, making the model less prone to deception. After that, in sub-task resolution stage, the LLM is prompted to provide the solutions for every sub-task. Finally, in the solution merge stage, the LLM is prompted to assembly the solutions of sub-tasks and acquire the final answer.To tackle tasks of different sizes, Divide-and-Conquer prompting strategy can be divided to two variants: Single-Level DaC Solver and Multi-Level DaC Solver.

Algorithm 1 Single-Level Divide-and-Conquer Solver $T(S, a, t, L, f)$

Require: Input Sequence S, Prompt m (for solution merge), Prompt t (for sub-task tackling), Prompt d (for task decomposition), LLM L
Ensure: Results of the task on input sequence S
1: $\{S_1, S_2, ..., S_k\} \leftarrow L(d, S)$
2: Result $\leftarrow \emptyset$
3: **for** $i = 1, 2, ..., k$ **do**
4: Result $\leftarrow$ Result $+[SEP] + L(t, S_i)$
5: **end for**
6: **Return** $L(m, Result)$

Single-level Divide-and-Conquer Solver decomposes the task in one call to the LLM, which expands the original task as a tree of one level. The algorithm is presented in the Alg. 1. The advantage of this variant is its simplicity and efficiency. However, when the original input is too long, single-level Divide-and-Conquer Solver may acquire sub-tasks with large problem sizes that will still trigger intermediate errors. In such a case, following (Khot et al., 2022), we can recursively expand the task as a multi-level tree. More specifically, we repeat the aforementioned steps to further divide the sub-tasks hierarchically until they are easy enough to be handled by the LLM. This can be done through a recursion program as presented in Alg. 2. More discussions on the proposed method's appliance scope, including its comparison with other prompting strategies and limitations, can be found in A.1

Algorithm 2 Multi-Level Divide-and-Conquer Solver Recursion $T(S, m, t, d, f, n, L)$

Require: Input Sequence S, Problem Size Metric Function $f(\cdot)$ (a function that measure the problem size), hyper-parameter w, Prompt m (for merge), Prompt t (for sub-task tackling), Prompt d (for task decomposition), Large Language Model L

Ensure: Results of the task on input sequence S

$S_1, S_2, ..., S_k \leftarrow L(d, S)$
Result $\leftarrow \emptyset$
for $i = 1, 2, ..., k$ **do**
    if $f(S_i) > w$ **then**
        Result $\leftarrow$ Result $+[SEP] + T(S_i, m, t, d, f, w, L)$
    else
        Result $\leftarrow$ Result $+[SEP] + L(t, S_i)$
    end if
end for
Return $L(m, Result)$

4 Theoretic Analysis

In this section, we provide theoretic analysis to the utility and limitations of the Divide-and-Conquer prompting. In the first subsection, we provide a comparison of IO prompting (common fixed-length instructional prompting) and DaC prompting in expressive power perspective. This part answers the first research question: the expressive power of IO prompting is a subset of DaC prompting. In the second subsection, we provide a comparison between Chain-of-Thoughts and DaC prompting in expressive power. Our comparison suggests that, although the expressive power of DaC prompting is a subset of Chain-of-Thoughts, for tasks satisfying specific conditions, DaC prompting can solve the problem with lower average context window length when decoding the tokens. Such property is empirically proved to be helpful for reducing the intermediate error and thus boost the performance.

4.1 Divide-and-Conquer vs. IO Prompting

We show that the expressive power of Divide-and-Conquer is stronger than IO Prompting:

Theorem 4.1 *We denote the set of problems that a fixed-precision transformer with fixed-length IO prompting can tackle as $S(IO)$. Similarly, we denote the set of problems that a fixed-precision transformer with DaC prompting can tackle as $S(DaC)$. Then we have the following results:*

$$S(IO) \subset \mathsf{TC}^0 \subseteq \mathsf{NC}^1 \subseteq S(DaC) \tag{1}$$

Proof Sketch: The conclusion that $S(IO) \subset \mathsf{TC}^0$ is a corollary of the main results in (Chiang et al., 2023). In this paper, we mainly focus on proving $\mathsf{NC}^1 \subseteq S(DaC)$. Specifically, we exploit 2-color Binary Subtree Isomorphism (2-BSI) problem for the proof. In (Jenner et al., 2003), 2-BSI problem is proved to be an NC^1-complete problem. Its definition is:

Definition 1 ***2-color Binary Subtree Isomorphism problem*** *is that, given a pattern 2-colorbinary tree t_p and a base 2-color binary tree t_b, a solver is required to judge whether the pattern tree is isomorphic to a sub-tree of t_b*

In (Jenner et al., 2003), the authors pointed out that the encoding of the problem will influence the hardness of the problem. In this paper, we focus on pointer list encoding of 2-BSI. Detailed information about the pointer list encoding of 2-BSI can be found in Appendix. For pointer list encoding of 2-BSI, we have the following theorem:

Theorem 4.2 *There exists a log-precision transformer with fixed depth L and hidden dimension d that can solve the 2-BSI of any size with fixed-length prompt m (for merge), t (for sub-task tackling) and d (for task decomposition).*

Proof Sketch: The detailed proof is provided in the Appendix A.2. Here we give a brief flow of the proof. To prove this theorem, we first show an algorithm that can solve the problem with divide-and-conquer strategy. Then we prove that there exists a log-precision transformer with fixed depth L and hidden dimension d that can express the modules in the algorithms with different but fixed-length prompts. In this way, we can prove the theorem.

With the above theorem, we can prove that $\mathsf{NC}^1 \subseteq S(DaC)$, which finishes the proof. With this theoretic results, we can answer the **RQ 1**:

Compared to IO prompting, DaC have theoretically guaranteed advantages in expressive power.

4.2 DaC vs. CoT

In this section, we compare Divide-and-Conquer with Chain-of-Thoughts in order to understand the utility and limitation of DaC prompting. The limitation of DaC prompting is that its expressive power is a subset of CoT prompting:

Proposition 4.3 *We denote the set of problems that a fixed-precision transformer with DaC prompting can tackle as $S(DaC)$. Similarly, we denote the set of problems that a fixed-precision transformer with CoT prompting can tackle as $S(CoT)$ Then we have the following results:*

$$S(DaC) \subseteq S(CoT) \tag{2}$$

The proof of this proposition is very straightforward. For any problem that DaC can solve, we can concatenate all outputs of LLM in dividing, tackling and merging as a sequence. Then we can prompt LLM with CoT to output this sequence. Thus, the problem set that DaC can resolve is a subset of CoT.

The limitation revealed by the above theorem shows that compared to CoT, the appliance scope of Divide-and-Conquer is limited. However, by analyzing the average decoding context window size, we show that on specific tasks, divide and conquer can reduce the problem complexity:

Definition 2 Decoding Context Window Size: *In auto-regressive decoding, each token is decoded from a window that covers all previous tokens. We denote the length of the window as the Decoding Context Window Size of the token.*

Proposition 4.4 *Suppose that a task contains k sub-tasks, each of which does not rely on the answers of other sub-tasks. We define such sub-tasks as* **parallel sub-tasks***. If an LLM tackle these sub-tasks sequentially with CoT, then the average decoding context window size of the sub-tasks' resolution will be $C + \frac{\sum_{i=1}^{k} r_i - 1}{2}$, where r_i is the length of the response to the i-th sub-task and C is the length of input context. If we tackle them parallely with DaC, then the average decoding context window size of the sub-tasks' resolution will be $C + \sum_{i=1}^{k} \frac{(r_i-1)^2}{2\sum_{j=1}^{k} r_j} < C + \frac{\sum_{i=1}^{k} r_i - 1}{2}$.*

The above proposition shows that when task contains a large amount of **parallel sub-tasks**, DaC is more helpful for reducing the average decoding context window size than CoT. Existing works have empirically showed that long decoding context window will propagate intermediate errors and thus increase the probability of generating hallucination (Yang & Ettinger, 2023). Thus, we can acquire a conclusion that DaC is competetive on tasks that contain a large amount of **parallel sub-tasks** and are bothered by intermediate errors and hallucination. With these theoretic results, we can answer the **RQ 2**:

Compared to CoT and its variants, DaC prompting's expressive power is weaker. However, on tasks containing a large amount of **parallel sub-tasks***, DaC is more helpful.*

4.3 ADVANTAGES OF DAC

The above analysis answer the two research questions that we proposed. By summarizing these two answers, we can acquire the two conditions such that when a task simultaneously satisfied both conditions, DaC bring performance boost:

- **Condition 1:** *the task is harder than $S(IO)$, such as TC^0-complete problems and NC^1-complete problems.*
- **Condition 2:** *the task contains a large amount of parallel sub-tasks and is bothered by hallucinations or intermediate errors.*

In Tab. 1, we present some sample tasks that satisfied the conditions. Also, we list some tasks that typically do not satisfy the conditions. This is helpful for guiding prompt engineering. Details are provided in Appendix A.6.

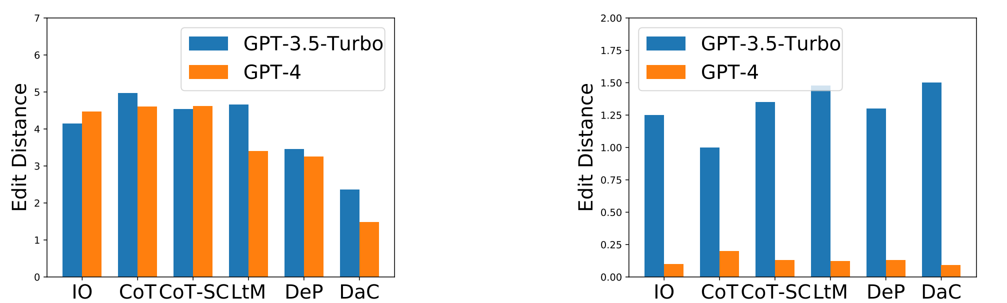
Applicable Tasks	Non-Applicable Tasks
Integer Multiplication	Integer Addition
Fact Verification	Multi-round QA
Consistency Evaluation	Planning

Table 1: We list some exaple tasks that satisfy the conditions and some tasks that do not satisfy the conditions.

5 EXPERIMENTS

5.1 CASE 1: LONG INTEGER ARITHMETIC

In this case, we consider two tasks in long integer arithmetic: **Multiplication**, which satisfy the conditions we proposed, and **Addition**, which does not satisfy the first condition [1]. Our experiment results will show that DaC prompting bring performance boost on multiplication and does not bring boost on integer addition.



(a) Edit distance of DaC and baseline prompting strategies on GPT-3.5 and GPT-4 for Multiplication.

(b) Edit distance of DaC and baseline prompting strategies on GPT-3.5 and GPT-4 for Addition.

Figure 3: Performance of different prompting strategies on long integer multiplication.

Task Setup: For this task, we randomly generated 200 pairs of 5-digit integers. We choose 5 for the digit length because according to previous works, ChatGPT-3.5 gets 0% accuracy on 4-digit multiplications (Cheng & Zhang, 2023), and ChatGPT-4 gets close to 0% accuracy on 5-digit multiplications (Yang et al., 2023). We evaluate the performance with Edit Distance (Marzal & Vidal, 1993; Schaeffer et al., 2023).

Setup of baselines and DaC: In this task, our baselines include IO prompting, Chain of Thought (CoT), CoT-SC, Least-to-Most (LtM), and Decomposed Prompting (DeP). Tree-of-Thoughts is not

[1] Multiplication is a TC^0-complete problem and can be divided to multiple parallel sub-tasks, while Addition is in $S(IO)$Barcelo et al. (2023)

24emStrategies	GPT-3.5-Turbo				GPT-4			
	F1	Acc	Prec	Recall	F1	Acc	Prec	Recall
IO-prompting	61.69	61.27	62.11	61.28	64.07	72.66	**93.41**	48.76
Chain-of-Thoughts	46.85	64.26	**91.36**	31.50	71.05	76.10	90.08	58.66
CoT-SC	47.70	64.25	88.83	32.60	71.39	76.36	90.41	58.98
Tree-of-Thoughts	70.40	59.91	55.83	**95.34**	69.41	71.73	75.53	64.28
Least-to-Most	56.43	64.91	74.42	45.44	72.51	77.11	90.74	60.38
Divide-and-Conquer	**74.84**	**75.55**	77.41	72.03	**76.92**	**78.99**	85.36	**70.01**

Table 2: Performance of different prompting methods on HaluEval dataset.

applicable. This is because that multiplication is deterministic calculation without requiring search in a tree. For DaC, we apply Multi-Level Divide-and-Conquer program-guided solver.

Results: Experimental results are shown in Fig. 3(a) and 3(b). As we can see, for integer addition which does not satisfy our proposed conditions, the performance of DaC, CoT and its variants does not significantly outperform IO prompting for both ChatGPT-3.5 and 4. However, for integer multiplication which satisfy our proposed conditions, under all settings, our proposed prompting strategy outperform all the baselines. This phenomenon indicate that our proposed conditions are useful for recognizing the tasks where DaC is more powerful.

5.2 Case 2: Fact Verification of Long Text

In the previous section, we show that for arithmetic tasks, our proposed conditions are discerning to the tasks where divide-and-conquer has advantages. In this section, we further present our conditions can be applied to natural language tasks. Specifically, we present the performance of baselines and Divide-and-Conquer on fact verification of long text. In this task, the LLM is required to whether a long corpus is aligned with base knowledge. This task **satisfied the proposed two conditions**. For the first condition, we can reduce a 2-BTI problem to fact verification by describing the two trees with natural language. In this way, we can convert the trees to two paragraphs and what we need to do is to ask the LLM to judge whether the two paragraphs are aligned or not. For the second condition, since we are tackling long text, then each sentence can be regarded as parallel sub-tasks. We select two benchmarks of fact verification: **Fact-Verification for Hallucination Detection** and **Fact-Verification for Misinformation Detection**

5.2.1 Hallucination Detection

Although Large Language Models have achieved impressive performance on various NLP tasks, they are bothered by hallucination problem (Manakul et al., 2023), especially when the generated content or the input context is too long for the user to have a thoroughly review (Zhang et al., 2023). In this paper, we focus on evaluating the performance of different strategies in guiding LLM to recognize inconsistency between given context and model response with hallucination.

24emStrategies	GPT-3.5-Turbo				GPT-4			
	F1	G-M	Prec	Recall	F1	G-M	Prec	Recall
Io-Prompting	72.12	72.77	83.22	63.64	69.15	71.77	94.44	54.55
Chain-of-Thoughts	56.09	60.64	90.48	40.64	74.03	75.79	94.21	60.96
CoT-SC	56.83	61.44	**91.67**	41.18	70.09	73.45	**100.0**	53.95
Tree-of-Thoughts	69.91	73.30	53.74	**100.0**	77.34	78.00	88.89	68.45
Least-to-Most	54.08	54.15	51.46	56.99	73.56	74.25	85.21	64.71
Divide-and-Conquer	**76.88**	**77.13**	83.65	71.12	**81.11**	**81.24**	76.67	**86.10**

Table 3: Performance of different prompting methods on SciFact dataset.

Task Setup: We use the HaluEval-Summary dataset. It is one of the three datasets in HaluEval benchmark for hallucination detection, which contains the hallucination generated by ChatGPT-3.5. HaluEval-Summary have the longest context and generated contents among all three tasks in this benchmark (Li et al., 2023a). Thus, detecting hallucination on this dataset requires repeatedly verify each sentence in the response, making standard prompting strategies acquire the worst accuracy across all three tasks. We report the Accuracy, F1 score (the hallucination pairs are positive samples), Precision and Recall.

Setup of baselines, ablation variants and DaC: In this task, our baselines include IO prompting, Chain of Thought, CoT-SC, Tree-of-Thoughts Least-to-Most, and Decomposed Prompting. In this task, the sub-tasks are verifying fragments of the summary, which are homogeneous and do not require recurssion. In such a setting, Decomposed Prompting is equivalent to LtM. For this task, we apply single level Divide-and-Conquer solver to decompose the summary to multiple sentences, handle them separately and then merge the conclusions of all sentences. The details are in Appendix.

Results: Experimental results are shown in Tab. 2. For both GPT-3.5 and GPT-4, our proposed prompting strategy outperform the baselines, presenting the advantage of DaC. More specifically, compared to IO-prompting, DaC achieved better performance in general, indicating the advantage brought by stronger expressive power. Meanwhile, compared to CoT and CoT-SC results, DaC clearly achieved much better recall. Tree-of-Thoughts, benefited by its searching ability, acquired significantly better recall score compared to other baselines. However, its significantly lower precision substantially harm its overall performance and leads to accuracy that is even worse than standard IO-prompting. In contrary, DaC carefully checked all sentences, locate the one containing factual error and merge the answers.

5.2.2 Misinformation Detection

The increasing abuse of misinformation toward manipulating public opinions on social media has been observed in different areas, such as healthcare (e.g. the recent COVID-19 pandemic) (Sharma et al., 2020; 2022). This threat is increasingly serious due to LLM's capacity in content generation (Li et al., 2023b; Weidinger et al., 2021; Zhang et al., 2022). This challenge raise the importance of fact-verification, which aims at judging the authenticity of an article based on a collection of evidence from verified source (Whitehouse et al., 2022; Zhang & Gao, 2023). In this experiment, we present that DaC can outperform other baselines in fact-verification involved with news article .

Task Setup: In this experiment, we mainly adopt SciFact dataset (Wadden et al., 2020). In SciFact dataset, each sample is a pair of news and evidence, where the evidence is the abstract of a peer-reviewed paper and the news is a sentence of claim. To better simulate the real-world scenario where news on social media usually appears as an paragraph of post, following Chen & Shu, we generate a dataset of paragraph-level misinformation based on SciFact dataset. Specifically, for a given claim, we apply ChatGPT-4 to extend the claim as an article based on the evidence. For this task, similar as hallucination detection, we apply single level Divide-and-Conquer solver to decompose the news article to multiple sentences, handle them separately and then merge the conclusions of all sentences. Also, the baselines in this experiments are the same as Hallucination Detection. The evaluation metrics includes F1 score, G-Mean score (geometric mean of precision and recall), Precision and Recall. We do not apply accuracy as the positive and negative classes are not balanced.

Results: Experimental results are shown in Tab. 3. Notably, GPT-3.5 incorporated with our proposed prompting strategy even outperform the performance of GPT-4 incorporated with IO-prompting, Least-to-Most, CoT and CoT-SC, which have significantly lower recall scores, indicating their proneness to deception. Only Tree-of-Thoughts, which is benefited by its advantage in exploring various options, acquired the best results among all baselines, but is still defeated by DaC. Moreover, as we can see, for GPT-4 the performance of CoT-SC is even worse than CoT, which is supposed to be a specific case of CoT-SC without exploration. These results suggests that, when facing deceptive contents generated on purpose, existing works' improvement may not be robust.

6 Conclusions

In this paper, we analyze the utility and limitations of divide-and-conquer prompting strategy. We first provide theoretic analysis to Divide-and-Conquer prompting and compare it with representative prompting strategies. Based on these theoretic results, we summarize two conditions under which a task is suitable for Divide-and-Conquer prompting. After that we conducted experiments on all several tasks. The empirical results validated our theoretic analysis and shows that the two conditions we proposed are helpful for recognizing the appliance scope of Divide-and-Conquer prompting.

REFERENCES

Pablo Barcelo, Alexander Kozachinskiy, Anthony Widjaja Lin, and Vladimir Podolskii. Logical languages accepted by transformer encoders with hard attention, 2023.

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

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

Canyu Chen and Kai Shu. Can llm-generated misinformation be detected? *arXiv preprint arXiv:2309.13788*, 2023.

Yuyan Chen, Qiang Fu, Yichen Yuan, Zhihao Wen, Ge Fan, Dayiheng Liu, Dongmei Zhang, Zhixu Li, and Yanghua Xiao. Hallucination detection: Robustly discerning reliable answers in large language models. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pp. 245–255, 2023a.

Zhikai Chen, Haitao Mao, Hang Li, Wei Jin, Hongzhi Wen, Xiaochi Wei, Shuaiqiang Wang, Dawei Yin, Wenqi Fan, Hui Liu, et al. Exploring the potential of large language models (llms) in learning on graphs. *arXiv preprint arXiv:2307.03393*, 2023b.

Vincent Cheng and Yu Zhang. Analyzing ChatGPT's mathematical deficiencies: Insights and contributions. In Jheng-Long Wu and Ming-Hsiang Su (eds.), *Proceedings of the 35th Conference on Computational Linguistics and Speech Processing (ROCLING 2023)*, pp. 188–193, Taipei City, Taiwan, October 2023. The Association for Computational Linguistics and Chinese Language Processing (ACLCLP). URL `https://aclanthology.org/2023.rocling-1.22`.

David Chiang, Peter Cholak, and Anand Pillay. Tighter bounds on the expressivity of transformer encoders, 2023.

Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. 2022.

Corinna Cortes and Neil D Lawrence. Inconsistency in conference peer review: revisiting the 2014 neurips experiment. *arXiv preprint arXiv:2109.09774*, 2021.

Antonia Creswell and Murray Shanahan. Faithful reasoning using large language models. *arXiv preprint arXiv:2208.14271*, 2022.

Wendi Cui, Jiaxin Zhang, Zhuohang Li, Damien Lopez, Kamalika Das, Bradley Malin, and Sricharan Kumar. A divide-conquer-reasoning approach to consistency evaluation and improvement in blackbox large language models. In *Socially Responsible Language Modelling Research*, 2023.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Sean Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, et al. Faith and fate: Limits of transformers on compositionality. *Advances in Neural Information Processing Systems*, 36, 2024.

Guhao Feng, Yuntian Gu, Bohang Zhang, Haotian Ye, Di He, and Liwei Wang. Towards revealing the mystery behind chain of thought: a theoretical perspective. *arXiv preprint arXiv:2305.15408*, 2023.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International Conference on Machine Learning*, pp. 10764–10799. PMLR, 2023.

Jose A Garcia, Rosa Rodriguez-Sánchez, and J Fdez-Valdivia. Quality censoring in peer review. *Scientometrics*, 126:825–830, 2021.

Amr Hendy, Mohamed Abdelrehim, Amr Sharaf, Vikas Raunak, Mohamed Gabr, Hitokazu Matsushita, Young Jin Kim, Mohamed Afify, and Hany Hassan Awadalla. How good are gpt models at machine translation? a comprehensive evaluation. *arXiv preprint arXiv:2302.09210*, 2023.

Beizhe Hu, Qiang Sheng, Juan Cao, Yuhui Shi, Yang Li, Danding Wang, and Peng Qi. Bad actor, good advisor: Exploring the role of large language models in fake news detection. *arXiv preprint arXiv:2309.12247*, 2023.

Birgit Jenner, Johannes Köbler, Pierre McKenzie, and Jacobo Torán. Completeness results for graph isomorphism. *Journal of Computer and System Sciences*, 66(3):549–566, 2003.

Jaehun Jung, Lianhui Qin, Sean Welleck, Faeze Brahman, Chandra Bhagavatula, Ronan Le Bras, and Yejin Choi. Maieutic prompting: Logically consistent reasoning with recursive explanations. *arXiv preprint arXiv:2205.11822*, 2022.

Daniel Kahneman. *Thinking, fast and slow*. macmillan, 2011.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks. *arXiv preprint arXiv:2210.02406*, 2022.

Junyi Li, Xiaoxue Cheng, Wayne Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. Halueval: A large-scale hallucination evaluation benchmark for large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 6449–6464, 2023a.

Siyu Li, Jin Yang, and Kui Zhao. Are you in a masquerade? exploring the behavior and impact of large language model driven social bots in online social networks. *arXiv preprint arXiv:2307.10337*, 2023b.

Yang Liu, Yuanshun Yao, Jean-Francois Ton, Xiaoying Zhang, Ruocheng Guo Hao Cheng, Yegor Klochkov, Muhammad Faaiz Taufiq, and Hang Li. Trustworthy llms: a survey and guideline for evaluating large language models' alignment. *arXiv preprint arXiv:2308.05374*, 2023.

Potsawee Manakul, Adian Liusie, and Mark JF Gales. Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models. *arXiv preprint arXiv:2303.08896*, 2023.

Rui Mao, Guanyi Chen, Xulang Zhang, Frank Guerin, and Erik Cambria. Gpteval: A survey on assessments of chatgpt and gpt-4. *arXiv preprint arXiv:2308.12488*, 2023.

Andres Marzal and Enrique Vidal. Computation of normalized edit distance and applications. *IEEE transactions on pattern analysis and machine intelligence*, 15(9):926–932, 1993.

William Merrill and Ashish Sabharwal. The parallelism tradeoff: Limitations of log-precision transformers. *Transactions of the Association for Computational Linguistics*, 11:531–545, 2023a. doi: 10.1162/tacl_a_00562. URL `https://aclanthology.org/2023.tacl-1.31`.

William Merrill and Ashish Sabharwal. The expresssive power of transformers with chain of thought. *arXiv preprint arXiv:2310.07923*, 2023b.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners.

Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo. Are emergent abilities of large language models a mirage? *arXiv preprint arXiv:2304.15004*, 2023.

Karishma Sharma, Sungyong Seo, Chuizheng Meng, Sirisha Rambhatla, and Yan Liu. Coronavirus on social media: Analyzing misinformation in twitter conversations. *arXiv preprint arXiv:2003.12309*, 2020.

Karishma Sharma, Yizhou Zhang, and Yan Liu. Covid-19 vaccine misinformation campaigns and social media narratives. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 16, pp. 920–931, 2022.

Yiming Tan, Dehai Min, Yu Li, Wenbo Li, Nan Hu, Yongrui Chen, and Guilin Qi. Evaluation of chatgpt as a question answering system for answering complex questions. *arXiv preprint arXiv:2303.07992*, 2023.

Jonathan P Tennant and Tony Ross-Hellauer. The limitations to our understanding of peer review. *Research integrity and peer review*, 5(1):6, 2020.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

David Wadden, Shanchuan Lin, Kyle Lo, Lucy Lu Wang, Madeleine van Zuylen, Arman Cohan, and Hannaneh Hajishirzi. Fact or fiction: Verifying scientific claims. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 7534–7550, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.609. URL `https://aclanthology.org/2020.emnlp-main.609`.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

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

Laura Weidinger, John Mellor, Maribeth Rauh, Conor Griffin, Jonathan Uesato, Po-Sen Huang, Myra Cheng, Mia Glaese, Borja Balle, Atoosa Kasirzadeh, et al. Ethical and social risks of harm from language models. *arXiv preprint arXiv:2112.04359*, 2021.

Chenxi Whitehouse, Tillman Weyde, Pranava Madhyastha, and Nikos Komninos. Evaluation of fake news detection with knowledge-enhanced language models. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 16, pp. 1425–1429, 2022.

Yuanhao Wu, Juno Zhu, Siliang Xu, Kashun Shum, Cheng Niu, Randy Zhong, Juntong Song, and Tong Zhang. Ragtruth: A hallucination corpus for developing trustworthy retrieval-augmented language models. *arXiv preprint arXiv:2401.00396*, 2023.

Chenghao Yang and Allyson Ettinger. Can you follow me? testing situational understanding for chatgpt. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 6385–6398, 2023.

Zhen Yang, Ming Ding, Qingsong Lv, Zhihuan Jiang, Zehai He, Yuyi Guo, Jinfeng Bai, and Jie Tang. Gpt can solve mathematical problems without a calculator. *arXiv preprint arXiv:2309.03241*, 2023.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023.

Xuan Zhang and Wei Gao. Towards llm-based fact verification on news claims with a hierarchical step-by-step prompting method. *arXiv preprint arXiv:2310.00305*, 2023.

Yizhou Zhang, Defu Cao, and Yan Liu. Counterfactual neural temporal point process for estimating causal influence of misinformation on social media. *Advances in Neural Information Processing Systems*, 35:10643–10655, 2022.

Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, et al. Siren's song in the ai ocean: A survey on hallucination in large language models. *arXiv preprint arXiv:2309.01219*, 2023.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.

Xinyu Zhu, Junjie Wang, Lin Zhang, Yuxiang Zhang, Ruyi Gan, Jiaxing Zhang, and Yujiu Yang. Solving math word problem via cooperative reasoning induced language models. *arXiv preprint arXiv:2210.16257*, 2022.

Mingyu Zong and Bhaskar Krishnamachari. Solving math word problems concerning systems of equations with gpt-3. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 15972–15979, 2023.

A Appendix

A.1 Discussions and Limitations

In summary, the proposed method has following advantages:

Comparison with IO-Prompting: Superiority in Expressive Power As we proved in Sec. 4, Compared to IO-prompting, DaC has stronger expressive power and thus can solve harder problems.

Comparison with CoT and EoT: Disentangling the task decomposition and task resolution Compared to the prompting family of CoT and EoT, DaC explicitly separate the task decomposition stage and task resolution stage. Therefore, we can acquire explicit decomposed sub-task rather than intermediate thoughts proposed during decoding. Consequently, we can explicitly enumerate all sub-tasks output by the decomposition module and avoid the model from missing important sub-tasks.

Comparison with LtM and Decomposed Prompting: Parallel Sub-task Handler and Sequential Sub-task Handler Similar as DaC, some program-guided prompting like LtM and Decomposed Prompting also explicitly separate the task decomposition stage and task resolution stage. However, they are mainly designed for multi-step reasoning for complex tasks. Thus, they sequentially tackle the sub-tasks and assembly the resolutions. As a result, they tend to follow the flow of the deceptive contents, leading to proneness to deceptive content.

Although DaC DaC surpasses the baselines on the proposed tasks, it still has some **limitations**. The first issue is that the appliance scope of DaC is still limited. More specifically, CoT, EoT, LtM and DaC are based on different algorithmic paradigms, learning to different Appliance Scopes. As pointed out by Feng et al., CoT and LtM can be considered as a neural **dynamic programming** algorithm. Thus, CoT is more suitable for tasks that can be bridged to dynamic programming, such as multi-step question answering. Differently, EoT is based on **exploration and search**, which is more suitable for planning and search, such as Game of 24 (Yao et al., 2023). DaC is based on **Divide-and-Conquer algorithm**. Thus, it is more suitable for tasks that can be decomposed to a series sub-tasks that are disjoint or only slightly overlapped. Our future work will focus on further expand the appliance scope of DaC to more areas like question answering.

A.2 Proof to Theorem 4.2

Before providing the proof, we first formally define how to organize the inputs (i.e., two 2-color trees) as a sequence. We assume that we acquire two trees t_p of size n and t_b of size m. They are organized as two sequences of nodes with a random order. Each node has three variables: color, left child index, and right child index. If any child is null, then the index is filled with 0. Then we can organize them as as two sequences $\mathbf{X}_p \in \mathbb{R}^{n\times 3}$ and $\mathbf{X}_b \in \mathbb{R}^{n'\times 3}$, where each item in the sequence is a vector of 3 dimensions. The first dimension is the index of the left child, the second dimension is the index of the right child, the third dimension is the color indicator (0 or 1). In addition, we have a root vector $\mathbf{r}$ with three dimensions. The first dimension is the index of the root node of t_p (i.e., pointing to the root node of t_p) and the second is the index of the root node of t_b (i.e., pointing to the root node of t_b). The third dimension of $\mathbf{r}$ is filled with 0 to make it have same dimension as the items in $\mathbf{X}_p$ and $\mathbf{X}_b$. This expression of trees is also called as pointer list encoding according to (Jenner et al., 2003). Note that in the following proof, we assume that all indices start from 1. Thus 0 is regarded as a NULL pointer.

Following the proof flow we provided in Sec. 4.2, we first provide the following divide-and-conquer algorithm that can solve the above problem:

The algorithm described above is a typical divide-and-conquer algorithm for solving rooted tree isomorphism. Its justification can be found in many textbooks introducing algorithms, such as *Introduction to Algorithms* (Cormen et al., 2022). Here we provide the detailed definition and implementation of problem size metric $f(\cdot)$, hyper-parameter w, merge function $m()$, sub-task tackling function $t(\cdot)$, task decomposition function $d(\cdot)$:

- $w = 1$, and $f(\mathbf{r}, \mathbf{X}_p, \mathbf{X}_b)$ is defined as the depth of the pattern tree t_p indicated with root vector $\mathbf{r}$. Although precisely calculating $f(\mathbf{r}, \mathbf{X}_p, \mathbf{X}_b)$ is of $O(n)$, judging whether

Algorithm 3 Recursion Divide-and-Conquer Algorithm for 2-BSI $BSI(\mathbf{r}, \mathbf{X}_p, \mathbf{X}_b, m, t, d, f, w)$

Require: Inputs $\mathbf{r}, \mathbf{X}_p, \mathbf{X}_b$, problem size metric function $f(\cdot)$, hyper-parameter w, merge function m, sub-task tackling function t, task decomposition function d

Ensure: A 0-1 indicator vector $\mathbf{v}$: if there exists a subtree with node i as root that is isomorphic with pattern tree t_p defined with inputs $\mathbf{r}, \mathbf{X}_p, \mathbf{X}_b$, then the $\mathbf{v}[i]$ is 1. Otherwise, $\mathbf{v}[i]$ is 0.

1: $\mathbf{r}_l, \mathbf{r}_r \leftarrow d(\mathbf{r}, \mathbf{X}_p, \mathbf{X}_b)$
2: **for** $i \in \{l, r\}$ **do**
3:     **if** $f(\mathbf{r}_i, \mathbf{X}_p, \mathbf{X}_b) > w$ **then**
4:         $\mathbf{v}_i \leftarrow BSI(\mathbf{r}_i, \mathbf{X}_p, \mathbf{X}_b, m, t, d, f, w)$
5:     **else**
6:         $\mathbf{v}_i \leftarrow t(\mathbf{r}_i, \mathbf{X}_p, \mathbf{X}_b)$
7:     **end if**
8: **end for**
9: **Return** $m(\mathbf{r}, \mathbf{X}_p, \mathbf{X}_b, \mathbf{v}_l, \mathbf{v}_r)$

Algorithm 4 Implementation of $d(\mathbf{r}, \mathbf{X}_p, \mathbf{X}_b)$ when the depth of the tree indicated by $\mathbf{r}$ is not longer than 2

Require: Inputs $\mathbf{r} \in \mathbb{R}^3, \mathbf{X}_p \in \mathbb{R}^{n \times 3}, \mathbf{X}_b \in \mathbb{R}^{n' \times 3}$

Ensure: A 0-1 indicator vector $\mathbf{v}$: if there exists a subtree with node i as root that is isomorphic with pattern tree t_p defined with inputs $\mathbf{r}, \mathbf{X}_p, \mathbf{X}_b$, then the $\mathbf{v}[i]$ is 1. Otherwise, $\mathbf{v}[i]$ is 0.

1: $\mathbf{r}_l \leftarrow <\mathbf{X}_p[\mathbf{r}[1], 2], \mathbf{r}[2], \mathbf{r}[3]>$
2: $\mathbf{r}_r \leftarrow <\mathbf{X}_p[\mathbf{r}[1], 3], \mathbf{r}[2], \mathbf{r}[3]>$
3: **Return** $\mathbf{r}_l, \mathbf{r}_r$

Algorithm 5 Implementation of $t(\mathbf{r}, \mathbf{X}_p, \mathbf{X}_b)$ when the depth of the tree indicated by $\mathbf{r}$ is not longer than 2

Require: Inputs $\mathbf{r} \in \mathbb{R}^3, \mathbf{X}_p \in \mathbb{R}^{n \times 3}, \mathbf{X}_b \in \mathbb{R}^{n' \times 3}$

Ensure: A 0-1 indicator vector $\mathbf{v}$: if there exists a subtree with node i as root that is isomorphic with pattern tree t_p defined with inputs $\mathbf{r}, \mathbf{X}_p, \mathbf{X}_b$, then the $\mathbf{v}[i]$ is 1. Otherwise, $\mathbf{v}[i]$ is 0.

1: Initialize $\mathbf{v}$ as all q vector with a length of n'
2: **if** $\mathbf{r}[1] == 0$ **then**
3:     **Return** $\mathbf{v}$
4: **end if**
5: **for** $i \in \{1, 2, ..., m\}$ **do**
6:     **if** $\mathbf{X}_b[i, 3]! = \mathbf{X}_p[\mathbf{r}[1], 3]$ **then**
7:         $\mathbf{v}[i] \leftarrow 0$
8:     **end if**
9: **end for**
10: **Return** $\mathbf{v}$

$f(\mathbf{r}, \mathbf{X}_p, \mathbf{X}_b) > 1$ only require us to check whether the root node has child. If not, then return False.

- $d(\mathbf{r}, \mathbf{X}_p, \mathbf{X}_b) = \mathbf{r}_l, \mathbf{r}_r$ returns two new root vectors $\mathbf{r}_l, \mathbf{r}_r$. Both r_l, r_r have the same second and third dimension as $\mathbf{r}$. The $\mathbf{r}_l$'s first dimension is updated to be the index of the left child of the root node that $\mathbf{r}$ points to. The $\mathbf{r}_r$'s first dimension is updated to be the index of the right child of the root node that $\mathbf{r}$ points to. The updating function can be written as:
- $t(\mathbf{r}, \mathbf{X}_p, \mathbf{X}_b) = \mathbf{v}$ returns a 0-1 indicator vector $\mathbf{v} \in \mathbb{R}^m$ with the same length of the base tree size. If there exists a subtree with node i as root that is isomorphic with pattern tree t_p defined with inputs $\mathbf{r}, \mathbf{X}_p, \mathbf{X}_b$, then the $\mathbf{v}[i]$ is 1. Otherwise, $\mathbf{v}[i]$ is 0. When the pattern tree's depth is not higher than 1 (i.e., 1-node tree), $t(\mathbf{r}, \mathbf{X}_p, \mathbf{X}_b)$ is equivalent to output a 0-1 vector indicating the nodes in the base tree that have the same color of the root node of pattern tree. The implementation is provided in Alg. 5.
- $m(\mathbf{r}, \mathbf{X}_p, \mathbf{X}_b, \mathbf{v}_l, \mathbf{v}_l) = \mathbf{v}$ merge the results $\mathbf{v}_l, \mathbf{v}_l$ to acquire a 0-1 indicator vector $\mathbf{v} \in \mathbb{R}^m$ with the same length of the base tree size. If there exists a subtree with node i as root that is isomorphic with pattern tree t_p defined with inputs $\mathbf{r}, \mathbf{X}_p, \mathbf{X}_b$, then the $\mathbf{v}[i]$ is 1. Otherwise,

Algorithm 6 Implementation of $m(\mathbf{r}, \mathbf{X}_p, \mathbf{X}_b, \mathbf{v}_l, \mathbf{v}_r)$

Require: Inputs $\mathbf{r} \in \mathbb{R}^3, \mathbf{X}_p \in \mathbb{R}^{n \times 3}, \mathbf{X}_b \in \mathbb{R}^{n' \times 3}, \mathbf{v}_l \in \mathbb{R}^n, \mathbf{v}_r \in \mathbb{R}^n$

Ensure: A 0-1 indicator vector $\mathbf{v}$: if there exists a subtree with node i as root that is isomorphic with pattern tree t_p defined with inputs $\mathbf{r}, \mathbf{X}_p, \mathbf{X}_b$, then the $\mathbf{v}[i]$ is 1. Otherwise, $\mathbf{v}[i]$ is 0.

Initialize $\mathbf{v}$ as all 0 vector with a length of n'
if $\mathbf{r}[1] == 0$ **then**
    Return $\mathbf{v}$
end if
for $i \in \{1, 2, ..., m\}$ **do**
    if $\mathbf{X}_b[i, 3] == \mathbf{X}_p[\mathbf{r}[1], 3]$ **then**
        if $\mathbf{v}_l[\mathbf{X}_b[i, 1]]] == 1$ and $\mathbf{v}_r[\mathbf{X}_b[i, 2]]] == 1$ **then**
            $\mathbf{v}[i] \leftarrow 1$
        else if $\mathbf{v}_l[\mathbf{X}_b[i, 2]]] == 1$ and $\mathbf{v}_r[\mathbf{X}_b[i, 1]]] == 1$ **then**
            $\mathbf{v}[i] \leftarrow 1$
        end if
    end if
end for
Return $\mathbf{v}$

$\mathbf{v}[i]$ is 0. This function can be implemented by checking whether the pattern root's children have a perfect match with each node's children. Since each node has at most two children, checking the perfect match can be done in constant time. The implementation is provided in Alg. 6.

After providing the detailed implementation of the functions $d(\cdot), t(\cdot), m(\cdot)$, we are going to prove that there exists one unified transformer that can handle all these tasks with different prompts d, t, m. First, we will provide the following Lemma:

Lemma A.1 *Any fixed-size logic circuit that only contains multi-fan-in AND gates, multi-fan-in OR gates, NOT gates and has no recurrent structure can be precisely simulated by a multi-layer perceptron (MLP) with ReLU activation function and a width of $O(|Input| + |Circuit|)$ and a depth of $O(|Circuit|)$, where $|Input|$ denotes the size of input and $|Circuit|$ denotes the number of gates in the circuit.*

Assume that we are given a series of input pins with logic variable of 0 or 1, organized as a 0-1 vector $\mathbf{x} \in \mathbb{R}^h$. We first prove that all gates can be simulated by a two-layer perceptron. Then we can serialize all gates in the circuits and stack their corresponding 2-layer simulators accordingly to acquire a MLP simulator. An AND gate that take $\mathbf{x}$ as input can be simulated as:

$$\text{AND}(\mathbf{x}) = \sigma(w_A \mathbf{x} - h + 1) \tag{3}$$

where σ is the ReLU activation function, and w_A is a weight vector with all dimensions equal to 1. If some dimensions of $\mathbf{x}$ are not the input of the gate, we can set the corresponding dimensions in the weight vector as 0 and adjust the h as the input pin number. Similarly, an OR gate that take $\mathbf{x}$ as input can be simulated as:

$$\text{OR}(\mathbf{x}) = 1 - \sigma(w_O \mathbf{x} + h + 1) \tag{4}$$

where σ is the ReLU activation function, and w_O is a weight vector with all dimensions equal to -1. A NOT gate is different, since it only takes one input pin. In such a case, we denote the index of the input pin as i, then we can simulate a NOT gate as:

$$\text{NOT}(\mathbf{x}) = \sigma(w_N \mathbf{x} + 1) \tag{5}$$

where w_N is a weight is a weight vector whose i-th dimension equals to -1 and all other dimensions equal to 0. Also, since the x is a 0-1 vector, the activation function is equivalent to a identical function to x:

$$\mathbf{x} = \sigma(\mathbf{x}) \tag{6}$$

To construct a MLP that can simulate a fixed-size logic circuit without recurrent structure, we apply the circuit serialization in (Merrill & Sabharwal, 2023b) which order the gates based on topological order. In this way, we can represent the circuit as a sequence GATE[1], GATE[2], GATE[3],...,GATE[L], where each GATE[i]'s input only contains the output of the previous gates and the original input $\mathbf{x}$. Therefore, we can construct a $2L$-layer MLP base on the above serialization. Specifically, the $2i$-th and $2i+1$-th layers of the MLP will simulate the $GATE[i]$ as well as copy all previous inputs with activation function and concatenate them together. This can be done by concatenate an identical matrix on the GATE's weight vector (w_A, w_O or w_N). In this way, we can construct a MLP that precisely simulate the circuit. Since every time we concatenate the output of a gate with the input of it, the input dimension number of the final layer can be bounded by $O(|\mathbf{x}| + L)$. In the worst case, for a circuit of size L, we needs $2L$ layers to precisely simulate it. However, in many cases, a lot of gates in the circuits can be run parallelly. In such cases, the MLP could be much more shallow.

Now, we can start to prove our main theorem:

Theorem A.2 *There exists a log-precision transformer with fixed depth and hidden dimension that can solve the 2-BSI of any size with fixed-length prompt m (for merge), t (for sub-task tackling) and d (for task decomposition).*

We prove this theorem by constructing a Transformer that can tackle this problem. First we define how to organize the input given $\mathbf{r}, \mathbf{X_p}, \mathbf{X_b}$ and the prompt. Specifically, we construct a feature sequence $\mathbf{X} \in \mathbb{R}^{(3+n+n')\times 7}$. Each item in this sequence is a feature of 7 dimensions, indicating a token. The first two dimensions indicate whether the token is a prompt ('00'), a root vector ('01'), a pattern tree node ('10'), or a base tree node ('11'). The third to fifth dimensions carries the information about the token. For a prompt token, '100' indicates merge prompt m, '010' indicates sub-task tackling prompt t, and '001' indicates task decomposition prompt d. For other cases, these three dimensions are with the same formula as the three dimensions in $\mathbf{r}, \mathbf{X}_p, \mathbf{X}_b$. The rest two dimensions are allocated specifically for the merge function $m(\cdot)$ to store $\mathbf{v}_l$ and $\mathbf{v}_r$. More specifically, for the feature of token indicating the i-th base tree node, its sixth dimension is $\mathbf{v}_l[i]$ and its seventh dimension is $\mathbf{v}_r[i]$. For other tokens, these two dimensions are filled with 0. In $\mathbf{X}[1]$ we store the prompt token. In $\mathbf{X}[2]$ and $\mathbf{X}[3]$ we store the input root vector $\mathbf{r}$ duplicately. We store the same token twice so that we can tackle $\mathbf{r}_l$ and $\mathbf{r}_r$ separately. To separate this two token, we use the last dimension, which was padded as 0 in $\mathbf{r}$, to distinguish them. $\mathbf{X}[2,5]$ is set as 0 and $\mathbf{X}[3,5]$ is set as 1. From $\mathbf{X}[4]$ to $\mathbf{X}[3+n]$, we store $\mathbf{X}_p$. From $\mathbf{X}[4+n]$ to $\mathbf{X}[3+n+n']$, we store $\mathbf{X}_b$. For all node indices of pattern tree, we add them by 3. For all node indices of base tree, we add them by 3+n, so that the indices can be applied to directly retrieve the positional embeddings. After preparing the inputs, we start to construct our Transformer. The transformer first attach the position index for each token (positional embedding). After that, the inputs are forwarded into a transformer with depth of 2. Each transformer layer contains a multi-head attention layer followed by a MLP. As proved by (Merrill & Sabharwal, 2023b; Feng et al., 2023), the attention layer of Transformer can retrieve the feature of tokens whose positional embeddings satisfy specific conditions. For multi-head attention, different heads can retrieve tokens with different conditions. In the following construction, we will use this conclusion to construct attention heads with different functions.

In the first Transformer layer, the function of each attention head is defined as:

- Head 1 only attends to the token itself to store $X[i]$ for token i.
- Head 2 attends to the token with a positional embedding matches the $\mathbf{X}[i,3]$ and copy this token's 5-dimension feature. For tree node tokens, this head's job is to retrieve the feature of $\mathbf{X}[i]$'s left child. For root vector tokens, this head's job is to retrieve the feature of pattern tree root node. For the first token (prompt token), this head's retrieved feature will not be applied in the afterwards layers and thus does not influence the correctness of the model.
- Similar as Head 2, Head 3 attends to the token with a positional embedding matches the $\mathbf{X}[i,4]$ and copy this token's 5-dimension feature. This head's job is to retrieve the feature of $\mathbf{X}[i]$'s right child. For root vector tokens, this head's job is to retrieve the feature of base tree root node.
- Head 4 attends to the first token (prompt token) and copy this token's 7-dimension feature. This head's job is to retrieve the prompt indicator.
- Head 5 attends to the second token (root token) and copy this token's 7-dimension feature. This head's job is to retrieve the root information.

Algorithm 7 Logic circuit for MLP of the second Transformer layer

Require: Input feature $\mathbf{x}'' \in \mathbb{R}^{42}$
Ensure: Output feature $\mathbf{y} \in \mathbb{R}^{7}$

$\mathbf{y} \leftarrow \mathbf{x}''[1:7]$ {Initialize $\mathbf{y}$}
if $\mathbf{x}''[1:2] == 00$ or $\mathbf{x}''[1:2] == 10${Prompt Token or Pattern Tree Node} **then**
  Return $\mathbf{y}$
else if $\mathbf{x}''[1:2] == 01$ {Root Vector Token} **then**
  if $\mathbf{x}''[24:26] == 001${Prompt is d} **then**
    if $\mathbf{x}''[5] == 0$ **then**
      $\mathbf{y}[3] \leftarrow \mathbf{x}''[10]$ {get $\mathbf{r}_l$, similar as line 1 in Alg. 4}
    else if $\mathbf{x}''[5] == 1$ **then**
      $\mathbf{y}[3] \leftarrow \mathbf{x}''[11]$ {get $\mathbf{r}_r$, similar as line 2 in Alg. 4}
    end if
  end if
else if $\mathbf{x}''[1:2] == 11$ {Base Tree Node Token} **then**
  if $\mathbf{x}''[24:26] == 010${Prompt is t} **then**
    if $\mathbf{x}''[40] == \mathbf{x}''[5]${Line 6 in Alg. 5} **then**
      $\mathbf{y}[5] \leftarrow 1$
    else
      $\mathbf{y}[5] \leftarrow 0$
    end if
  else if $\mathbf{x}''[24:26] == 100${Prompt is m} **then**
    if $\mathbf{x}''[13] == 1$ and $\mathbf{x}''[21] == 1$ {Line 7 in Alg. 6} **then**
      $\mathbf{y}[5] \leftarrow 1$
    else if $\mathbf{x}''[14] == 1$ and $\mathbf{x}''[20] == 1${Line 9 in Alg. 6} **then**
      $\mathbf{y}[5] \leftarrow 1$
    else
      $\mathbf{y}[5] \leftarrow 0$
    end if
  end if
end if

With the above 5 heads, the attention layer will output a 35-dimension feature for each token. We denote these features as $\mathbf{X}' \in \mathbb{R}^{(3+n+n')\times 35}$. After that, these features are forwarded into a MLP fitting identical mapping to acquire the input features for the second Transformer layer.

In the second Transformer layer, the function of each attention head is defined as:

- Head 1 only attends to the token itself to store $X'[i]$ for token i.
- Head 2 attends to the token with a positional embedding matches the $\mathbf{X}'[i, 31]$ and copy this token's 1-7 dimension features ($\mathbf{X}'[\mathbf{X}'[i, 31], 1:7]$). This head's job is to broadcast the feature of the pattern tree root node to every token.

With the above 2 heads, the attention layer will output a 42-dimension feature for each token. We denote these features as $\mathbf{X}'' \in \mathbb{R}^{(3+n+n')\times 42}$. For root vector token, only the features from head 1 and head 4 are useful. For base tree node tokens, all 42 dimensions are useful. Then each token's feature are parallely forwarded into a MLP. We will use this MLP to fit the logical circuit described in Alg. 7. The function of Alg. 7 is to aggregate the functions of $m(\cdot), t(\cdot), d(\cdot)$ together and assign the correct value based on the prompt indicator. In Alg. 7, all operations are AND, OR, NOT, SELECTOR, and ASSIGN and there is not loop. Thus, it is a static logical circuit and can be implemented with multi-fan-in AND, OR, NOT gates. Thus, it can be precisely simulated by a MLP according to our Lemma A.1.

After acquiring the $\mathbf{y} \in \mathbb{R}^7$ for each token, we can organize them as a feature sequence $\mathbf{Y} \in \mathbb{R}^{(3+n+n')\times 7}$. When the prompt is d, we return $\mathbf{Y}[2, 3:5]$ as $\mathbf{r}_l$ and $\mathbf{Y}[3, 3:5]$ as $\mathbf{r}_r$. If the prompt is t or m, then we can output $\mathbf{Y}[3+n+1 : 3+n+n', 5]$ as the expected $\mathbf{v}$.

A.3 JUSTIFICATION TO PROPOSITION 4.4

Suppose that the LLM is auto-regressively decoding n tokens from an input context window with length of C. Then the decoding window of the i-th token is $C + i - 1$. Thus, the average window size will be:

$$\frac{\sum_{i=1}^{n}(C + i - 1)}{n} = \frac{C + n - 1}{2} \tag{7}$$

Thus, when we sequentially decode all the sub-task resolutions, the total length of the decoded sequence will be $\sum_{i=1}^{k} r_i$. Thus the average window size will be:

$$C + \frac{\sum_{i=1}^{k} r_i - 1}{2} \tag{8}$$

Meanwhile, when we apply Divide-and-Conquer, we parallely decode each sub-task's resolution. Thus, for each sub-task, total window size will be $C\sum_{j=1}^{k} r_j + \sum_{i=1}^{k} \frac{(r_i-1)r_i}{2}$. Thus the average window size will be $C + \sum_{i=1}^{k} \frac{(r_i-1)r_i}{2\sum_{j=1}^{k} r_j}$. Meanwhile, with Jensen inequility, we have:

$$\sum_{i=1}^{k}(r_i - 1)r_i < \sum_{i=1}^{k}(r_i - 0.5)^2 \le (\sum_{i=1}^{k}(r_i - 0.5))^2 \le (\sum_{i=1}^{k} r_i - 0.5k)^2 \tag{9}$$

Thus, when $k \ge 2$, we have:

$$\sum_{i=1}^{k}(r_i - 1)r_i < (\sum_{i=1}^{k} r_i - 1)^2 \tag{10}$$

Thus, we have:

$$C + \sum_{i=1}^{k} \frac{(r_i - 1)^2}{2\sum_{j=1}^{k} r_j} < C + \frac{\sum_{i=1}^{k} r_i - 1}{2} \tag{11}$$

A.4 PROMPTING DETAILS OF DAC

Multiplication of Long Integers: Suppose we have two $2n$-digit numbers AB and CD, where A, B, C, D are all n-digit numbers. Then we can break $AB \times CD$ as $(A \times C \times 10^{2n}) + (A \times D \times 10^{n}) + (B \times C \times 10^{n}) + (B \times D)$, where the calculation in each bracket pair is disjoint with others bracket pairs. We only need to compute the results of multiplication in each bracket pair parallelly and then merge all of them with addition:

Decomposer Prompt d: Please split the string a from the middle as two separated strings. The lengths of the two separated strings should be as close as possible. Please only return the two strings separated by a comma and do not return anything else.

Sub-task Tackling Prompt t: (1)Please compute $a * b$. (2) Please only return the final results and do not return anything else (ensure disentangled-sub-process principle).

Merge Prompt m: Please compute $x = a * 10^{2n} + b * 10^{n}$ and $y = c * 10^{n} + d$. Based on the above calculation, please compute $x + y$ carefully step by step.

Hallucination Detection in Long Context: We divide the summary to sentences. After that, we paralelly verify the sentences. Finally, we merge the verification to each sentence:

Decomposer Prompt d: Please help me segment the following paragraph as sentences. The separated sentence should be output as: #Statement 1#: ...#Statement 2#: ...Do not say anything else. Just return the statements in the given format.
Paragraph

Sub-task Tackling Prompt t: I want you to act as a factual contradiction checker. You are given a set of statements and a document. Among the statements, there might be one or more statement that contains contradictions with the document. Please find the problematic statement if it exist by analyzing the statements one by one. For each statement, please make a choice:

- A: The statement is totally aligned with the document for sure.

- B: The statement contradicts with the document.

Merge Prompt m: Based on the above analysis, please tell me, does any statement above contain contradiction with the document?.

Fact-Verification for Misinformation Detection: Similar as hallucination detection, we divide the summary to sentences. After that, we paralelly verify the sentences. Finally, we merge the verification to each sentence. Thus, our decomposer prompt and sub-task tackling prompt are the same as hallucination detection. The only difference is the merge prompt.

Merge Prompt m: If we connect the above statements to be a news article, based on the above analyzation, please answer me: Is there any contradiction between the document and the article?

A.5 Decomposed Prompting and Least to Most

Least-to-Most (LtM) Prompting (Zhou et al., 2022) and Decomposed Prompting (Khot et al., 2022) are two similar works to our work. They both propose to explicitly prompt the LLM to decompose the task as a series of sub-tasks and sequentially tackle them. In Fig .2, we merge these two methods. Here, we will provide more detailed comparison of them, which is shown in Fig. 4. Decomposed Prompting can regarded as a upgraded version of LtM. It introduces special notations into the prompt to represent program states so that when sequentially tackling the sub-tasks, it can call heterogeneous modules to tackle them. Such design enable the LLM to call external programs (e.g., retrieval documents on WikiPedia and program based calculator) and/or itself (i.e., recursion). Such design endows it stronger expressive power and increases the compositional generalization ability of LLMs in different areas, such as symbolic manipulation and multi-hop QA (Khot et al., 2022). Also, it endows LLM the ability to do open-domain QA by retrieving from external knowledge base.

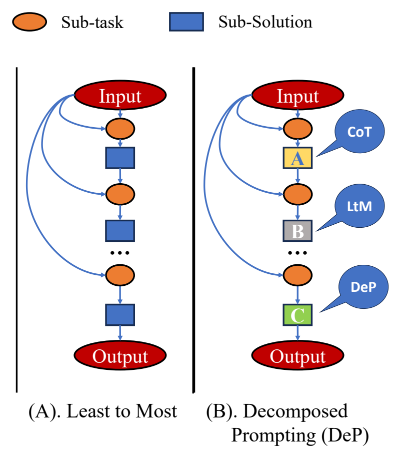


Figure 4: Comparison of Least-to-Most (LtM) Prompting and Decomposed Prompting (DeP).

A.6 Typical Tasks that Satisfy and Dissatisfy the Proposed Conditions

To better assist the prompt engineering on different tasks, we list the typical tasks that satisfy and dissatisfy the proposed conditions. In common tasks, the following tasks satisfy the proposed conditions. For such tasks, searching good decomposition prompt for DaC is likely to be helpful for the performance:

1. **Multiplication**
2. **Fact Verification on Long Text**
3. **Auto Evaluation on Long Text**
4. **Article-level Summary**

The following tasks typically do not satisfy the proposed conditions. For such tasks, searching good decomposition prompt for DaC is not very likely to be helpful for the performance:

1. **Addition**: It is too simple and violate the condition 1
2. **Division**: It does not contain parallel sub-tasks, thus violate condition 2

3. **Multi-Round Question-Answering**: It is a typical sequential task, thus violate condition 2
4. **Planning**: It is a typical sequential task, thus violate condition 2

A.7 More Discussions on Sequential Sub-task Tackling and Parallel Sub-task Tackling

Example of Sequential Sub-task Tackling

Complete Task: Compute 12345*67890:

Sub-task 1: Compute x=45*90:
A: ......

Sub-task 2: Based on the above result, compute y=123*90*10^2+45*90:
A: ......

Sub-task 3: Based on the above result, compute z=45*678*10^2+123*90*10^2+45*90:
A: ......

Sub-task 4: Based on the above result, compute w=123*678*10^4+45*678*10^2+123*90*10^2+45*90:
A: ......

Example of Parallel Sub-task Tackling

Complete Task: Compute 12345*67890:

Sub-task 1: Compute x=45*90:
A: ......

Sub-task 2: Compute y=123*90*10^2:
A: ......

Sub-task 3: Compute z=45*678*10^2:
A: ......

Sub-task 4: Compute w=123*678*10^4:
A: ......

Resolution Assembly: Based on the above computation, compute x+y+z+w
A: ......

Figure 5: Toy example of Sequential Sub-task Tackling and Parallel Sub-task Tackling in long integer multiplication

Example of Sequential Sub-task Tackling

Complete Task: Verify the following summary:
#Summary#: **A video** showing the final moments of Germanwings Flight 9525 has been recovered by investigators from the wreckage site. Marseille prosecutor Brice Robin urged anyone who might have more footage to turn it over immediately. Andreas Lubitz, the co-pilot accused of deliberately crashing the plane, had a history of severe depression and suicidal tendencies.

Sub-task 1: Verify the following statement:
#Summary#: **A video** showing the final moments of Germanwings Flight 9525 has been recovered by investigators from the wreckage site.
A: ......

Sub-task 2: Based on the above analysis, verify the following statement :
#Summary#: **A video** showing the final moments of Germanwings Flight 9525 has been recovered by investigators from the wreckage site. Marseille prosecutor Brice Robin urged anyone who might have more footage to turn it over immediately.
A: ......

Sub-task 3: Based on the above analysis, verify the following statement :
#Summary#: **A video** showing the final moments of Germanwings Flight 9525 has been recovered by investigators from the wreckage site. Marseille prosecutor Brice Robin urged anyone who might have more footage to turn it over immediately. Andreas Lubitz, the co-pilot accused of deliberately crashing the plane, had a history of severe depression and suicidal tendencies.
A: ......

Example of Parallel Sub-task Tackling

Complete Task: Verify the following summary:
#Summary#: **A video** showing the final moments of Germanwings Flight 9525 has been recovered by investigators from the wreckage site. Marseille prosecutor Brice Robin urged anyone who might have more footage to turn it over immediately. Andreas Lubitz, the co-pilot accused of deliberately crashing the plane, had a history of severe depression and suicidal tendencies.

Sub-task 1: Verify the following statement :
#Summary#: **A video** showing the final moments of Germanwings Flight 9525 has been recovered by investigators from the wreckage site.
A: ......

Sub-task 2: Verify the following statement :
#Summary#: Marseille prosecutor Brice Robin urged anyone who might have more footage to turn it over immediately.
A: ......

Sub-task 3: Verify the following statement :
#Summary#: Andreas Lubitz, the co-pilot accused of deliberately crashing the plane, had a history of severe depression and suicidal tendencies.
A: ......

Resolution Assembly: Given the above analysis, verify the summary that consist of the above three statements.
A: ......

Figure 6: Toy example of Sequential Sub-task Tackling and Parallel Sub-task Tackling in hallucination detection

Sequential Sub-task Tackling and Parallel Sub-task Tackling are two different paradigm in decomposing complex tasks as sub-task to tackle. The first one decompose a complex tasks as a series of sub-tasks. In this series, each sub-task relies on the previous one's output as input or context. The second one decompose a complex tasks as a set of sub-tasks, each of which does not rely on others. Two examples for multiplication and hallucination detection are provided in Fig 5 and 6