Algorithm of Thoughts: Enhancing Exploration of Ideas in Large Language Models

Bilgehan Sel [1] **Ahmad Al-Tawaha** [1] **Vanshaj Khattar** [1] **Ruoxi Jia** [1] **Ming Jin** [1]

Abstract

Current literature, aiming to surpass the "Chain-of-Thought" approach, often resorts to external modi operandi involving halting, modifying, and then resuming the generation process to boost Large Language Models' (LLMs) reasoning capacities. Due to their *myopic perspective*, they escalate the number of query requests, leading to increased costs, memory, and computational overheads. Addressing this, we propose the *Algorithm of Thoughts*—a novel strategy that propels LLMs through algorithmic reasoning pathways. By employing algorithmic examples fully in-context, this overarching view of the whole process exploits the innate recurrence dynamics of LLMs, expanding their idea exploration with merely one or a few queries. Our technique outperforms earlier single-query methods and even more recent multi-query strategies that employ an extensive tree search algorithms while using significantly fewer tokens. Intriguingly, our results suggest that instructing an LLM using an algorithm can lead to performance surpassing that of the algorithm itself, hinting at LLM's inherent ability to weave its intuition into optimized searches. We probe into the underpinnings of our method's efficacy and its nuances in application. The code and related content can be found in: algorithm-of-thoughts.github.io.

1. Introduction

Recent developments in large language models (Chowdhery et al., 2022; Thoppilan et al., 2022; Liu et al., 2023, *inter alia*) have spotlighted their efficacy in general problem solving (Huang & Chang, 2022; Suzgun et al., 2022), code generation (Chen et al., 2021; Austin et al., 2021), and instruction following (Ouyang et al., 2022; Bai et al., 2022). While early models relied on direct answer strategies (Brown et al., 2020), contemporary research has shifted towards linear reasoning paths (Wei et al., 2022b; Kojima et al., 2022; Zhang et al., 2022) by breaking problems into sub-tasks for solution discovery, or harnesses external mechanisms to alter token generation by changing the context (Zhou et al., 2022a; Drozdov et al., 2022; Yao et al., 2023).

Analogous to human cognition (Sloman, 1996; Kahneman, 2011), early LLM strategies seemed to emulate the instantaneous *System 1*, characterized by its impulsive decision-making. In contrast, more recent methodologies like chain-of-thought (CoT) (Wei et al., 2022b) and least-to-most prompting (L2M) (Zhou et al., 2022a; Drozdov et al., 2022) reflect the analytical nature of *System 2*. Notably, integrating intermediary reasoning steps has yielded improvements in arithmetic reasoning tasks (Srivastava et al., 2022; Liang et al., 2022).

However, as tasks shift towards deeper planning and extensive thought exploration, these methods appear restrictive. Although CoT integrated with Self-Consistency (CoT-SC) (Wang et al., 2022) enlists multiple LLM outputs for a consensus, the lack of meticulous evaluation can result in model misdirection. The "Tree of Thoughts" (Yao et al., 2023; Long, 2023) emerges as a notable solution. While one LLM is dedicated to idea generation, another steps in to assess the merit of these ideas, following a halting-assessment-resuming cycle. This iterative process, based on a tree search, has shown marked effectiveness, especially in tasks with a breadth of continuations. We see this progression as akin to humans employing tools to circumvent working memory limitations, serving as an external augmentation for LLMs (Mialon et al., 2023; Sel et al., 2023; Gu et al., 2024a).

On the flip side, this enhanced LLM approach is not without pitfalls. A prominent downside is the substantial surge in the number of queries and computational demands. Each query to online LLM APIs such as GPT-4—a focal point of our study—incurs a monetary expense (Chen et al., 2023) but also contributes to latency, a significant limitation especially critical in real-time applications. Cumulative de-

[1]Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, USA. Correspondence to: Bilgehan Sel <bsel@vt.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

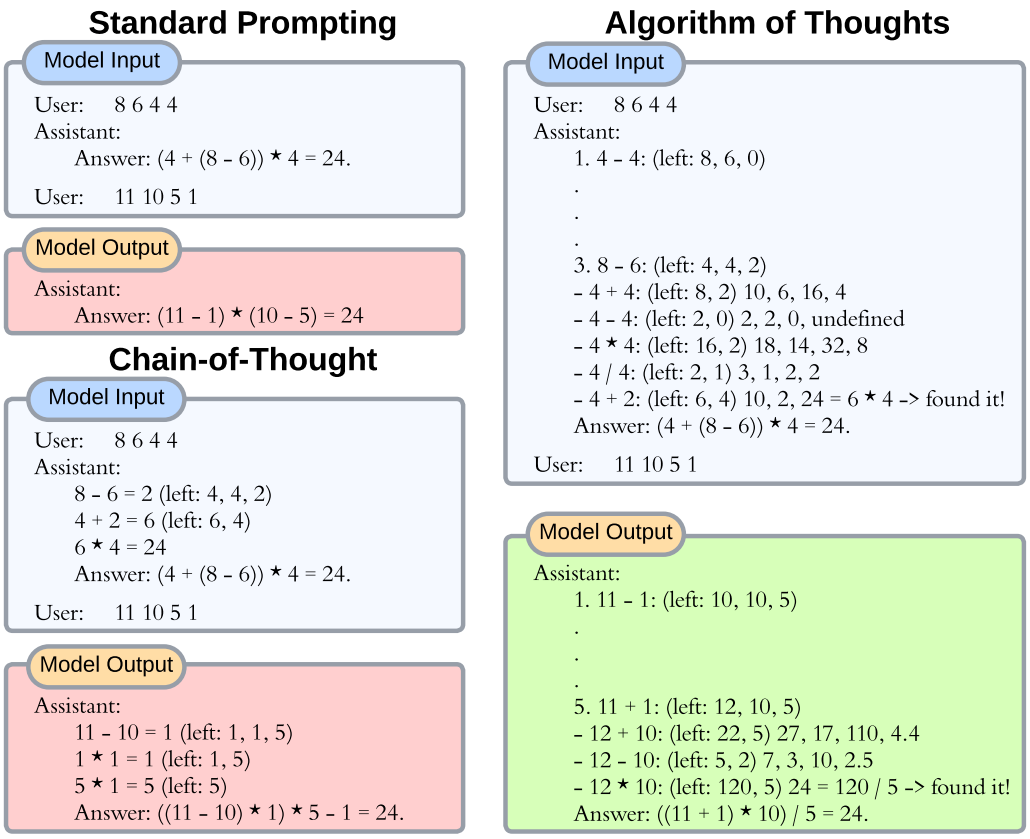

Figure 1. Comparison between standard prompting, CoT, and AoT in the game of 24. While standard prompting aims for a direct answer, CoT sketches out the successive steps to the final solution. AoT's in-context example, distinct from CoT, integrates the search process, highlighted by markers '1',..., '3' as "first operations" guiding subtree exploration for the problem set '8 6 4 4'. For clarity, only a single in-context example is displayed, with a focus on the third subtree exploration. AoT produces prospective search steps (e.g., the subtree exploration '5. 11 + 1') and evaluates potential subsequent steps to either progress towards a solution or retrace to another viable subtree.

lays from these queries can compromise solution efficiency. Infrastructure-wise, continuous interactions can stress systems, leading to potential bandwidth constraints and reduced model availability (Aminabadi et al., 2022). Moreover, the environmental implications cannot be ignored; incessant querying escalates the energy consumption of already power-hungry data centers, exacerbating the carbon footprint (Wu et al., 2022; Dhar, 2020; Khattar & Jin, 2023).

With this in mind, our goal was to *dramatically reduce the query counts employed by contemporary multi-query reasoning methods while maintaining performance for tasks necessitating adept use of world knowledge*, thereby steering a more responsible and proficient use of AI resources. Intriguingly, our aim has actually resulted in *surpassing* the performance of such techniques while *requiring significantly fewer tokens for prompting and generation.*

Reflecting on the evolution of LLMs from System 1 to System 2, an essential ingredient comes to light: algorithms (Sel et al., 2021; Al-Tawaha et al., 2023; Jin et al., 2023a; Gu et al., 2024b). Characterized by its methodical nature, the algorithmic perspective offers a path to keenly explore problem spaces, enact strategies, and formulate solutions (Al-Tawaha et al., 2021; Helie & Pizlo, 2022; Banerjee et al., 2022; Sel et al., 2022; Khattar et al., 2022). While much of the prevailing literature treats algorithms as *external* to LLMs (Lin et al.), given LLMs' inherent generative recurrence, can we channel this iterative logic to *internalize* an algorithm?

Drawing upon both the intricate nuances of human reasoning and the disciplined precision of algorithmic methodologies, our work aims to *fuse these two elements to enhance reasoning capabilities* within LLMs. Existing research underscores that humans, when navigating complex problems, instinctively draw upon past efforts, ensuring a comprehensive contemplation rather than a narrow focus (Monsell, 2003; Holyoak & Morrison, 2005; Baddeley, 2003). LLMs, with their generative span bounded only by token limits, appear poised to break through the barriers of human working memory. Spurred by this observation, we investigated if

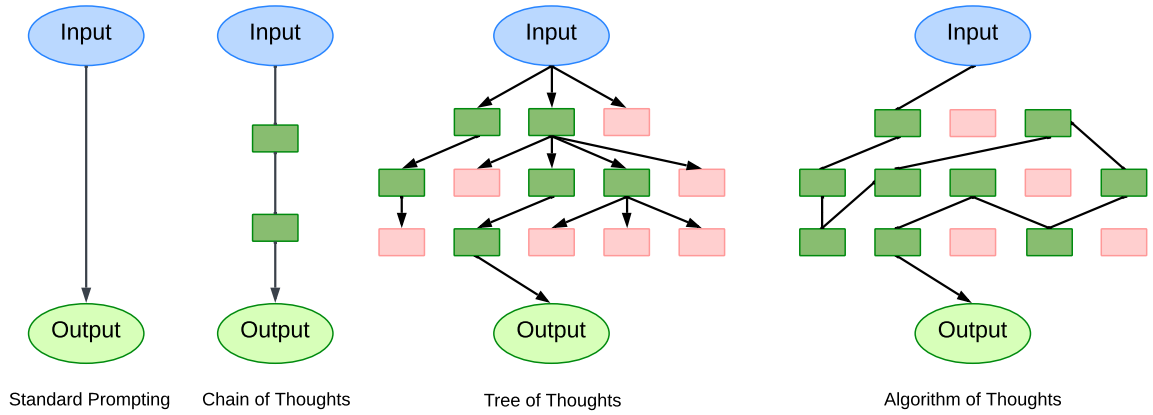

Figure 2. Illustration outlining various strategies for tackling reasoning problems with LLMs. Each box signifies a distinct thought, functioning as a unified string of words that forms an incremental pathway to reasoning. Green boxes indicate ideas deemed promising by the LLM, while red boxes represent less promising concepts.

LLMs could mirror a similar layered exploration of ideas, referencing prior intermediate steps to sieve out infeasible options, *all within their iterative generation cycle*. And while humans excel with their intuitive insight, algorithms stand out with organized, systematic exploration. Current techniques, like CoT, often sidestep this synergistic potential, imposing undue pressure on LLMs for on-the-spot precision. By capitalizing on LLMs' recursive capabilities, we emulate a hybrid human-algorithmic approach. This is achieved through our use of algorithmic examples that capture the essence of exploration, from initial candidates to validated solutions. Thus emerges our concept of the *Algorithm of Thoughts* (AoT), as illustrated in Figs. 1 and 2.

More broadly, our approach signifies a new paradigm of in-context learning. Instead of the traditional "supervised-learning" mold of [PROBLEM, SOLUTION] or [PROBLEM, SUCCESSIVE STEPS TO SOLUTION], we present a new structure that covers [PROBLEM, SEARCH PROCESS, SOLUTION]. Naturally, when instructing an LLM using an algorithm, the anticipation leans towards the LLM simply imitating the algorithm's iterative thinking. However, what emerges as intriguing is the LLM's ability to infuse its own "intuition" to achieve a search efficiency that even *surpasses* the algorithm itself (see Fig. 5).

In the subsequent sections, we first situate our work within the existing literature, followed by a discussion of our principal idea. We then present our experimental results and probe a series of hypotheses related to this emerging capability of LLM before rounding off with a conclusion.

2. Related Work

Standard Prompting. Also known as input-output prompting, it provides a few input-output examples of the task before getting an answer for the test sample from the language model (Brown et al., 2020). Although this method is very general and does not need any special prompting strategy, the performance is also worse compared to more advanced methods (Shao et al., 2023; Wei et al., 2022a; Lyu et al., 2023).

Chain-of-Thought. In CoT, LLMs are presented with examples where a given question x unfolds through a chain of intermediate reasoning pieces $c_1, \ldots, c_n$ to reach an answer y, represented as $x \rightarrow c_1 \rightarrow \ldots \rightarrow c_n \rightarrow y$ (Wei et al., 2022b; Lyu et al., 2023). By mimicking the examples in the context, the LLM automatically divides the solution into simpler linear steps to arrive at the answer, improving performance across numerous reasoning benchmarks. Self-consistency (Wang et al., 2022) is a widely used decoding strategy aimed at generating a variety of reasoning paths by choosing the final answer through a majority vote, though this necessitates additional generations. CoT can be further improved with integrating detailed algorithmic reasoning (Zhou et al., 2022b). We also utilize algorithmic examples in AoT, however, they are for emerging the inherent heuristic of LLMs to lead the search and not designed to follow a specified pseudocode, or are on language tasks, e.g., creative writing. Contrary to CoT's linear progression, our approach pivots towards the explorative aspect of LLMs. We reconceptualize the $c_1, \ldots, c_n$ sequence, not merely as successive steps towards a solution, but as a dynamic, potentially mutable path that resembles an algorithmic search, allowing for exploration, recalibration, and non-linear progression.

Least-to-Most prompting (L2M). Taking cues from educational psychology (Libby et al., 2008), L2M prompting directs the LLM to decompose the central problem into smaller subproblems. Each subproblem is tackled in sequence, with the outcome appended before progressing to the next (Zhou et al., 2022a; Drozdov et al., 2022). While this structured delineation is beneficial for broader generalization, it operates on the premise of finding a nearly perfect decomposition in a single attempt—ideal for problems with a clear-cut structure. Yet, when tasks intertwine with their decomposition complexities (like games of 24), this method's inflexibility becomes apparent. Contrastingly, AoT not only underscores the active subproblem (as shown in Fig. 1), but also permits a more contemplative approach by entertaining various options for each subproblem, while maintaining efficacy even with minimal prompts.

Tree of Thoughts (ToT). In the cases where each subproblem has multiple viable options to explore, linear reasoning paths from CoT or L2M substantially limit the coverage of the thought space. Considering possible options for each subproblem, the decision tree can be explored by external tree-search mechanisms (e.g., BFS, DFS) (Yao et al., 2023; Jin et al., 2023b; Sel et al., 2024). Evaluation capabilities of LLMs can also be used to direct the search by pruning nodes that are hopeless to increase efficiency. However, ToT, due to its requirement for multiple queries to the LLM for a solution, demands significantly more computation than AoT. Additionally, it necessitates evaluating the potential of each search node in the in-context examples and writing specialized functions to extract information from model responses to maintain the tree structure externally. In stark contrast, AoT requires just a single prompt and no coding skills, greatly democratizing LLM use for complex problems.

3. Algorithm of Thoughts

Our strategy pivots on recognizing *a core shortcoming* of current in-context learning paradigms. CoT, while enhancing the coherency of thought linkages leading to solutions, occasionally falters, presenting incorrect intermediate steps (Zelikman et al., 2022; Turpin et al., 2023; Lanham et al., 2023). Faithful CoT (Lyu et al., 2023) ought to amend this by eliciting symbolic chains of reasoning where the LLM's output resembles task-specific pseudo-code, primed for deterministic execution like Python. The intention is only to use the thought processes but not the outputs and inputs of each link since they have a tendency to be unreliable. But, *the occasional missteps of CoT may not necessarily be due to the LLM's inability to compute correctly*. The LLM, when confronted with questions that closely match conditions of previous in-context examples, may favor echoing those outputs over generating the appropriate questions. To shed light on this phenomenon, we designed an experiment.

Querying *text-davinci-003* for arithmetic tasks (e.g., '$11 - 2 =$'), we prefixed them with multiple in-context equations converging to an identical output (e.g. '$15-5 = 10, 8+2 = 10$'). Our results, presented in Fig. 3, reveal a steep decline in accuracy, suggesting that the mere presence of correct reasoning in the context might inadvertently compromise even basic arithmetic skills.

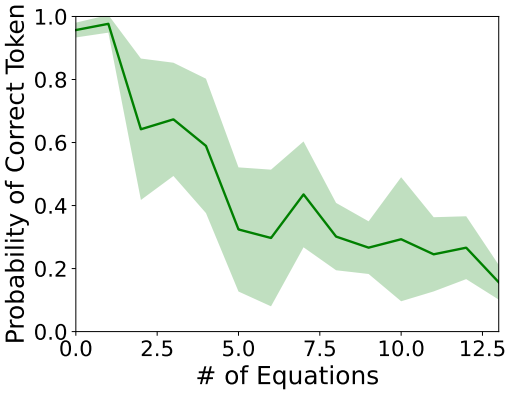

Figure 3. The probability of generating the correct token as we add more in-context examples that are correct but possess identical outputs.

To offset this bias, diversifying the outputs of examples might seem like a viable solution, but this could subtly skew the distribution of outputs. Merely adding unsuccessful trials, much like a random search, might inadvertently encourage the model to retry rather than truly solve. Capturing the true essence of algorithmic behavior, where both failed searches and subsequent recovering and learning from such attempts play a role, we incorporate *in-context examples patterned after search algorithms*, notably depth-first search (DFS) and breadth-first search (BFS). See Fig. 1 for an example.

This paper focuses on a broad class of tasks reminiscent of tree-search problems. These tasks necessitate breaking down the main problem, crafting feasible solutions for each segment, and making decisions on the paths to either pursue or forsake, with the option of reevaluating more promising segmentations. Rather than posing separate queries for every subset, we leverage the iterative capabilities of the LLM to address them in one unified generation sweep. By confining ourselves to one or two LLM interactions, this approach naturally incorporates insights from antecedent context candidates and tackles intricate issues requiring an in-depth exploration of the solution domain. We also give insights into how small or big those thoughts should be and what type of in-context examples should be given to the LLM to promote token efficiency. Subsequently, we

outline key components of tree-search algorithms and their manifestation in our framework.

1. Dividing the search into steps. Similar to creating step-by-step solutions in CoT or L2M, we also need to identify intermediate search layers. This is akin to creating examples for CoT, especially for tree-search problems, where the correct reasoning path resembles a CoT solution. The challenge lies in selecting the right chain from numerous candidates at each layer to reach the final answer. Thus, our focus will be more on generating the search process for in-context examples rather than how to solve each subproblem after selecting the next chain.

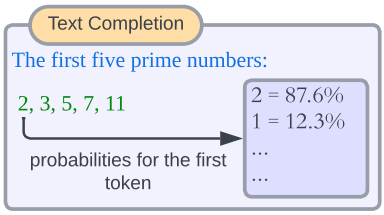

Figure 4. An example highlighting the drawback of isolated sampling of sequenced ideas. Input is denoted in blue, with the *text-davinci-003* providing the green completions.

2. Proposing Solutions to Subproblems. A dominant approach in existing works involves direct sampling from LLM token output probabilities (Wang et al., 2022; Yao et al., 2023). Though effective for one-off answers (Kadavath et al., 2022) , this method falls short in scenarios demanding a sequence of samples to be integrated or evaluated within subsequent prompts (Robinson & Wingate, 2022). To minimize model queries, we adopt an uninterrupted solution-creation process. Here, we directly and continuously generate solutions for the prevailing subproblem *without* any generation pauses.

The benefits are three-fold. First, with all generated solutions existing within a shared context, there is no need for individual model queries for each solution evaluation. Second, while it may seem counterintuitive, isolated token or token group probabilities might not always yield meaningful choices. A simple illustration is found in Fig. 4. When evaluated independently, the second-most probable token for our inaugural number is '1'—not qualifying as prime. But, when generation remains unbroken, the derived sequence is correct. This incongruence points towards the restrictive nature of the Markov property in sequence modeling. Core to our perspective is the premise that for sequential tasks like algorithmic search, LLMs are more adept at generating entire sequences than intermittently pausing and re-initiating the token sampling process.

3. Evaluating the Promise of a Subproblem. Existing techniques lean on additional prompting to discern the potential of tree nodes, aiding decisions regarding exploration direction. Our observations suggest that if the most promising routes are encapsulated within the in-context examples, LLMs inherently gravitate towards prioritizing those promising candidates. This diminishes the need for intricate prompt engineering and allows the incorporation of intricate heuristics, whether intuitive or knowledge-driven. Again, the absence of disjoint prompts in our approach allows for an immediate assessment of candidate viability in the same generation.

4. Backtracking to a More Promising Node. The decision of which node to explore next (including retracing to a prior node) inherently depends on the selected tree-search algorithm. While previous studies (Yao et al., 2023) have employed external means, such as coded mechanisms for the search process, this restricts its broader appeal and entails additional customization. Our designs predominantly adopt a DFS approach supplemented by pruning. The aim is to maintain proximity between nodes sharing the same parent, thereby encouraging the LLM to prioritize local over distant features. Additionally, we present performance metrics for the AoT approach grounded in BFS. Our reliance on the model's inherent capacity to glean insights from in-context examples obviates the necessity for additional mechanisms.

Expressiveness of LLMs with AoT. Recent works have investigated the expressivity of transformers with standard and CoT prompting (Chiang et al., 2023; Schuurmans, 2023; Merrill & Sabharwal, 2023; Feng et al., 2023). We provide the following theoretical result for AoT, which implies that it can tackle NP problems, extending from P problems that of COT's.

Corollary 3.1 (Informal)**.** *Consider* $\mathrm{TIME}(a^n)$ *as the class of problems for which a Turing machine exists that operates within a time complexity of* $\mathcal{O}(a^n)$ *for some* $a \geq 1$*. If a transformer can generate* a^n *intermediate tokens to solve the problem when prompted by AoT, we have*

$$\mathrm{TIME}(a^n) \subseteq \mathrm{AoT}(n), \tag{1}$$

where $\mathrm{AoT}(n)$ *refers to the decoding steps by AoT when the input has* n *tokens.*

The proof of the above corollary is given in the appendix.

4. Experiments

We show that AoT surpasses the performance of other single-prompt methods (e.g., standard, CoT/-SC prompting) and even that of strategies utilizing external mechanisms, such as ToT, across the benchmarks we tested. We present the

results for the creative writing task in the appendix. In addition, we show that AoT continues to have an advantage over standard prompting or CoT even after fine-tuning. This implies that the issue with LLMs is not simply a minor misalignment or a deficiency in domain expertise. Rather, it underscores the necessity of AoT prompting. This approach is vital because the nature of the tasks we evaluated inherently demands a thorough exploration of solution paths, a requirement that goes beyond simple fine-tuning adjustments. For the generation of in-context examples, we have asked the authors to write down their search process and randomly chosen from that list. We have written them again in a simple structured way to have uniformity between the examples to create our AoT prompts. More details regarding this process for each task is given in the AoT setup subsections.

4.1. Game of 24

The game of 24 is a mathematical card game in which players are given four numbers and must use addition, subtraction, multiplication, and division (each operation can be used more than once) to manipulate those numbers to reach a total of 24. For instance, for the numbers '8 8 5 4', one solution could be '$8 * (5 - (8/4)) = 24$'. At first glance, the game might appear straightforward. However, a cursory calculation suggests there are nearly 13,000 distinct expressions possible for any set of four numbers, making it a formidable challenge for present-day LLMs.

Task Setup. Adhering to the setup detailed in (Yao et al., 2023), we use games from indices 901-1000, sourced from the 1362 games ranked by relative difficulty at *4nums.com*. An attempt is considered successful if it is able to reach a total of 24 using the exact numbers provided and only the allowed operations.

Baselines. Standard prompting and CoT are used in the 5-shot setting, with CoT integrating 3 steps for the operations. These methods are sampled 100 times, and the averaged success rates from these samples are reported. CoT-SC is also tested with 100 votes in our setup. For ToT, we use a breadth of 5.

AoT Setup. We employ the same 5-shot setting as in standard prompting and CoT baseline setup. Our in-context samples leverage a DFS-style search algorithm, which is the same version used when contrasting with traditional DFS in Fig. 5. During each subtree exploration, dubbed either the 'first step' or 'first operation', we choose two numbers—illustrated by the selection of 8 and 6 in the third 'first step' (i.e., subtree labeled '3') of Fig. 1—and a corresponding operation (e.g., $8 - 6$). This operation results in a new number, 2, leaving us with three numbers in total. A thorough combing of these three numbers culminates in 19 leaf nodes, all visible under the '3' subtree in Fig. 1. In order to generate our in-context examples, we have randomly selected games that do not appear at test time. We asked the authors to write the search steps they used until they arrived at the answers. These are exactly the node selection, node expansion steps with inherent heuristics of the individuals. Then, we have selected randomly from these solutions and written them in a trivial structured way to assure uniformity between the examples. The exact prompts we use are given in the Prompts section under the 'AoT (DFS)' subsection in the appendix. We aim to assess two aspects: the ability of the LLM to pinpoint promising first operations, which directly impacts the number of resolved leaf nodes, and its performance against a conventional DFS. Details on the prompts are provided in the appendix. As our method emphasizes sequential generation over trajectory sampling, we operate with a temperature setting of 0.

Results. From Table 1, it is evident that standard prompting combined with CoT/-SC significantly lags behind tree search methods when used with LLMs. The "Standard + Refine" result, showing a 27% success rate, is referenced from (Yao et al., 2023). This method involves iteratively asking the LLM (up to 10 iterations) to refine its answer if the initial one is incorrect. Meanwhile, ToT is limited to a maximum of 100 node visits, translating to several hundred LLM queries for each example. Remarkably, AoT achieves its results with just a **single query!** Despite reducing the number of requests by more than a factor of 100, AoT still outperforms ToT in this task. Furthermore, AoT is also more efficient than ToT in terms of the total number of prompt tokens given to the LLM and the completion tokens it generates.

Method	Success	Queries	PTs	CTs
I/O	7.3%	1	164	18
CoT	4.0%	1	421	46.2
CoT-SC	9.0%	100	42,100	4,620
I/O + Refine	27%	10	458	360
ToT ($b = 5$)	69%	109.1	13,900	5,500
AoT (ours)	**71%**	1	5,450	998.4

Table 1. Game of 24: success rates and the average number of LLM queries for each example. We give the average query count, prompt tokens (PT), and completion tokens generated by the LLM (CT).

Error Analysis. Using a strictly LLM-centric approach—eschewing any external tooling or edits—we sought to categorize mistakes observed during the game of 24. This aids in highlighting areas for refinement when solely deploying LLMs. We've classified these errors into four distinct

categories: **1)** Out-of-token error: The LLM reaches its maximum token threshold without identifying a solution. **2)** Expression misstep: The LLM has the correct logic or steps but fails when trying to express or formulate them into a coherent answer. **3)** Non-finalization error: The LLM discovers the solution but continues its search without consolidating the finding. **4)** Other errors: This umbrella term encompasses other mistakes like computational errors that result in overlooking the solution or furnishing incorrect answers. To exclusively showcase the AoT's search capabilities, we also present the AoT + Manual Resolution version. Here, once the LLM pinpoints a solution, its final articulation is manually processed—a strategy also employed by the ToT method. As evidenced in Table 2, a notable 7% of mistakes stem from non-algorithmic factors like non-finalization and expression missteps. In fact, with manual resolution, AoT attains a 78% success rate, surpassing ToT. This underlines the potential for refining our prompt, especially in areas concerning recognizing and expressing successful problem resolutions. Additionally, the token limitation underscores the appeal of expanding the generative context window, which may further bolster LLMs' recursive reasoning when engaged with algorithmic examples.

Error Type	**Error**
Out-of-token error	9%
Expression misstep	4%
Non-finalization error	3%
Others	13%
Method	**Success**
ToT	69%
AoT	71%
AoT + Manual Resolution	78%

Table 2. Game of 24: AoT error analysis.

4.2. Mini Crosswords

The 5×5 mini crossword is a compact word puzzle featuring a grid of 25 squares arranged in a 5-by-5 configuration. Players are tasked with filling the grid based on provided clues for each word. Clues are given for words that run both across (horizontally) and down (vertically). Words intersect at certain letters, offering additional hints to complete the puzzle.

Task Setup. Adhering to the setup outlined in (Yao et al., 2023), we draw our prompts from games 136, 141, 146, 151, and 156 out of the 156 games available on *goobix.com*. Our testing focuses on a set of 20 games, specifically games 1, 6, . . ., 91, and 96.

Baselines. As done in the game of 24, we benchmark our method against established techniques: standard prompting, CoT, and ToT. For standard prompting, we provide both the crosswords and their respective solutions as in-context examples. CoT augments this by prompting the retrieval of words for each of the ten clues—equally split between horizontal and vertical orientations. We directly extract the success rates of ToT from their paper for comparison.

AoT Setup. We divide the process into two steps, each involving a query. Initially, we task the LLM with suggesting five potential words for each row and column. We then pinpoint the starting word candidates that have the highest compatibility with other words within the crossword framework. This preliminary phase mirrors a 'warm-up' sequence in algorithm initialization. In the subsequent step, we exclusively leverage the LLM's algorithmic reasoning prowess, starting with the pre-selected word. The method involves cyclically choosing a likely option for insertion, generating candidate words, and assessing their compatibility with the words already on the board. If no match is found, the process shifts focus to another promising candidate. Otherwise, the word is added to the crossword, and the search continues. The cycle concludes either when the board is fully populated or no more suitable words can be found, which may be due to either incorrect existing words or the absence of matching words. Notably, this entire process unfolds within a single-generation window. The algorithmic examples in our prompt (detailed in the Appendix) include three that achieve game completion and two that predominantly populate the crossword, filling 8 or 9 slots.

Results. Table 3 underscores AoT's proficiency in the mini crosswords task, showcasing a word success rate—a measure used in existing studies to represent the percentage of words correctly completed out of the total—that surpasses earlier methods reliant on various prompting techniques. It also outperforms ToT. An important observation is the sheer volume of queries ToT employs, exceeding AoT's by over a factor of 100. AoT also enjoys 25x reduction in total tokens required compared to ToT, a benefit of having everything in-context.

Method	**W. Success**	**Queries**	**PTs**	**CTs**
I/O	14%	1	790.3	30.5
CoT-SC	15.6%	1	1,400	1,600
ToT	46.5%	> 200	96,700	21.8k
AoT (ours)	**52%**	2	3,800	975.6

Table 3. 5×5 mini crosswords word: word success rates and the average number of LLM queries for each example. We give the average query count, prompt tokens (PT), and completion tokens generated by the LLM (CT).

Error Analysis. To understand the prevalent mistakes made by AoT, we've categorized the errors into four distinct categories. In our analysis for each game, we focus on the initial error the LLM produces while charting its reasoning path, given that an early error typically cascades into subsequent failures. **1)** No preselections: LLM fails to generate compatible words essential for the warm-start phase. Given a correctly preselected word, the second phase for recursive reasoning can exhibit errors including: **2)** Expression misstep: The LLM mistakenly believes it has exhausted all choices and jumps to an answer prematurely. **3)** Incorrect pattern extraction: The LLM wrongly extracts a pattern based on the current board layout. **4)** Erroneous word placement: Despite recognizing the correct pattern, the LLM selects a mismatched word or misses better-fitting alternatives. Navigating the crossword complexity arises from outdated terms and esoteric references. Predominantly, the errors observed are due to misguided word placements followed by pattern misinterpretations. Also, the LLM seems challenged in aligning letters at precise indices to create word structures— an obstacle circumvented by an external mechanism in the ToT framework.

Error Type	Error
No preselections	15.8%
Expression misstep	5.3%
Incorrect pattern extraction	26.3%
Erroneous word placement	52.6%

Table 4. Breakdown of errors in 5×5 mini crosswords with AoT. Numbers indicate the relative percentage of each error type among all errors.

4.3. Finetuning

In order to eliminate the possibility that prior experiments lacked domain knowledge or were misaligned with the task, even after few-shot prompting via standard prompting or CoT, we also finetuned GPT-3.5-Turbo using OpenAI's API with 900 examples with CoT and AoT. In Table 5, we can see that although GPT-3.5-Turbo had similar solution rates for the Game of 24 with CoT and AoT, AoT fine-tuning improved the model by 60% compared to 8% for CoT. This shows that fine-tuning alone cannot emerge implicit nonlinear thinking, and LLMs still require explicit exploration of possible options to arrive at a solution. This is similar to chess grandmasters being able to find better moves than others even when they play without thinking deeply. However, to find the truly great moves, they are also required to deliberately explore the possibility of space.

Method	w/o finetuning	w/ finetuning
CoT	3%	12%
AoT	3%	**63%**

Table 5. AoT's advantage continues even after finetuning. Finetuning results on the Game of 24 on 900 examples with CoT and AoT prompting.

5. Discussion

In this section, we delve into crucial aspects to consider when crafting prompts for AoT, using the game of 24 as our primary case study.

Can AoT surpass the DFS it is patterned after? A core query of ours is to ascertain if the LLM has the capability to not only mirror but also outdo the efficiency of the algorithm introduced in-context. As evidenced in Fig. 5, AoT systematically navigates fewer nodes than its DFS counterpart. While DFS employs a uniform strategy when choosing the subsequent subtree to investigate, AoT's LLM integrates its inherent heuristic. This amplification over the base algorithm exemplifies the advantages of LLM's recursive reasoning capability.

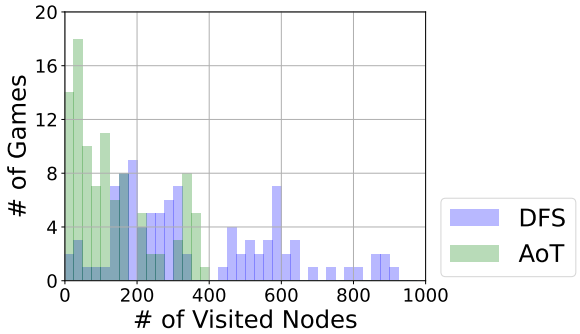

Figure 5. Histogram showing the number of visited nodes for AoT and DFS in the Game of 24.

How does the search step count within the algorithmic example modulate AoT's behavior? We begin with the standard AoT prompt and modify the subtree explorations. In AoT (Short), each in-context example uses one or two steps to reach a solution, while AoT (Long) incorporates three to five extra subtree explorations. The impact on total search steps is illustrated in Fig. 6. Our observations highlight longer generations for AoT (Long) and shorter ones for AoT (Short) relative to the original AoT. This suggests that the search step count introduces an implicit bias on the LLM's search velocity. Notably, even when navigating incorrect steps, it's essential to emphasize the exploration of promising directions.

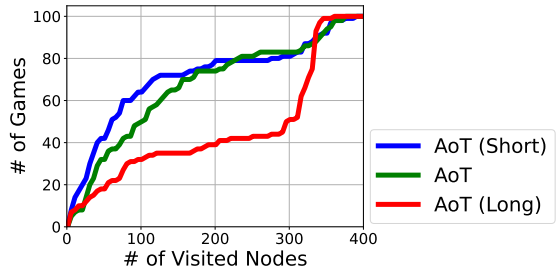

Figure 6. Comparison of AoT with shorter and longer in-context examples prompted AoT versions: cumulative number of games for the number of visited nodes.

Can AoT be used for question-answering tasks? To answer this question, we have followed the same structure to the Creative Writing task (given in the appendix of our paper) to evaluate AoT and the baselines on the first 100 questions of well-known GSM8K and StrategyQA benchmarks. Briefly, we implemented a zero-shot AoT prompt for StrategyQA and GSM8K that proposes 3 strategies and expands them with detail to select the best one. As seen in Table 6, we see a slight boost on this task due to GPT-4 with CoT already being competent.

Method	GSM8K	StrategyQA
IO	51%	73%
CoT	86%	82%
ToT	90%	83%
AoT	89%	84%

Table 6. Performance comparison of different methods on question-answer tasks using GSM8K and StrategyQA benchmarks. The AoT model shows competitive performance, especially when compared with the CoT and ToT methods.

Can AoT work as other dynamic programming methods? We have also tested AoT and the baselines on the traditional dynamic programming problems Coin Change and Edit Distance, where DFS and BFS have explosive complexities. However, another DP method named tabulation can more easily solve these problems. Since ToT cannot take the form of tabulation, and has to either DFS or BFS, we decided not to include those poor results to be fair. However, one can use a single leaf node with a CoT prompt to solve these problems. There, ToT's performance can be considered the same as that of CoT's. Please refer to Table 7 for the detailed results.

Can AoT help other SOTA LLMs? We investigate AoT for other SOTA LLMs, Claude 3 and Gemini 1.5 Pro, to see if it provides a significant boost for them as well on the game of 24. We see that both Claude 3 and Gemini 1.5 Pro benefit significantly. We were unable to run Gemini 1.5 Pro on CoT-SC due to API access not being available yet. Please refer to Table 8 for detailed results.

Problem	Coin Change	Edit Distance
I/O	72%	61%
CoT	76%	64%
AoT	96%	90%

Table 7. Performance comparison of AoT with traditional dynamic programming methods on solving Coin Change and Edit Distance problems.

Method	GPT-4	Claude 3	Gemini 1.5 Pro
IO	7%	6%	6%
CoT-SC	9%	9%	-
AoT	71%	68%	55%

Table 8. Additional language model results for the AoT, CoT-SC, and IO methods across different models.

6. Conclusion

This paper presents the *Algorithm of Thoughts*, a pioneering prompting strategy to navigate reasoning pathways in LLMs using minimal queries. Our findings reveal that this method not only substantially surpasses prior single-query techniques but also outperforms external tree-search implementations. Such an approach augments the potential to streamline idea discovery in LLMs, balancing both cost and computational demands. Future work includes designing token-efficient algorithmic examples, developing adaptive mechanisms for "tunnel-vision" activation to expedite the search, and deepening the understanding of this fresh mode of in-context learning from theoretical angles.

7. Limitations

While AoT substantially cuts down on the number of queries relative to ToT, its resource demands exceed those of standard prompting and CoT, a consequence of its extensive exploration of ideas via token generation. Crafting token-efficient algorithmic examples is one direction of future research. It is also pertinent to highlight that we conducted our tests exclusively with GPT-4. Though more costly than other LLMs, GPT-4's advanced capabilities appear pivotal for AoT's optimal functioning; models of lesser caliber might not yield comparable performance boosts from AoT.

Acknowledgments

This work was supported in part by the Amazon Research and Virginia Tech Initiative for Efficient and Robust Machine Learning and the National Science Foundation (Grants #2331775 and #2312794).

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Al-Tawaha, A., Kaushik, H., Sel, B., Jia, R., and Jin, M. Decision-focused learning for inverse noncooperative games: Generalization bounds and convergence analysis. *IFAC-PapersOnLine*, 56(2):9336–9341, 2023.

Al-Tawaha, A. S., Aljanaideh, K., and Alshorman, A. A singular value thresholding algorithm for order estimation. In *2021 American Control Conference (ACC)*, pp. 4478–4483. IEEE, 2021.

Aminabadi, R. Y., Rajbhandari, S., Awan, A. A., Li, C., Li, D., Zheng, E., Ruwase, O., Smith, S., Zhang, M., Rasley, J., et al. Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–15. IEEE, 2022.

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Baddeley, A. Working memory: looking back and looking forward. *Nature reviews neuroscience*, 4(10):829–839, 2003.

Bai, Y., Kadavath, S., Kundu, S., Askell, A., Kernion, J., Jones, A., Chen, A., Goldie, A., Mirhoseini, A., McKinnon, C., et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.

Banerjee, S., Bringsjord, S., Giancola, M., and Govindarajulu, N. S. Qualitative mechanical problem-solving by artificial agents:: Further progress, under psychometric ai. In *The International FLAIRS Conference Proceedings*, volume 35, 2022.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Chen, L., Zaharia, M., and Zou, J. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Chiang, D., Cholak, P., and Pillay, A. Tighter bounds on the expressivity of transformer encoders. *arXiv preprint arXiv:2301.10743*, 2023.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

Dhar, P. The carbon impact of artificial intelligence. *Nat. Mach. Intell.*, 2(8):423–425, 2020.

Drozdov, A., Schärli, N., Akyürek, E., Scales, N., Song, X., Chen, X., Bousquet, O., and Zhou, D. Compositional Semantic Parsing with Large Language Models. September 2022. URL https://openreview.net/forum?id=gJW8hSGBys8.

Feng, G., Gu, Y., Zhang, B., Ye, H., He, D., and Wang, L. Towards revealing the mystery behind chain of thought: a theoretical perspective. *arXiv preprint arXiv:2305.15408*, 2023.

Gu, S., Sel, B., Ding, Y., Wang, L., Lin, Q., Jin, M., and Knoll, A. Balance reward and safety optimization for safe reinforcement learning: A perspective of gradient manipulation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 21099–21106, 2024a.

Gu, S., Sel, B., Ding, Y., Wang, L., Lin, Q., Knoll, A., and Jin, M. Safe and balanced: A framework for constrained multi-objective reinforcement learning. *arXiv preprint arXiv:2405.16390*, 2024b.

Helie, S. and Pizlo, Z. When is psychology research useful in artificial intelligence? a case for reducing computational complexity in problem solving. *Topics in Cognitive Science*, 14(4):687–701, 2022.

Holyoak, K. J. and Morrison, R. G. *The Cambridge handbook of thinking and reasoning*. Cambridge University Press, 2005.

Huang, J. and Chang, K. C.-C. Towards reasoning in large language models: A survey. *arXiv preprint arXiv:2212.10403*, 2022.

Jin, M., Khattar, V., Kaushik, H., Sel, B., and Jia, R. On solution functions of optimization: Universal approximation and covering number bounds. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 8123–8131, 2023a.

Jin, M., Sel, B., Hardeep, F., and Yin, W. A human-on-the-loop optimization autoformalism approach for sustainability. *arXiv preprint arXiv:2308.10380*, 2023b.

Kadavath, S., Conerly, T., Askell, A., Henighan, T., Drain, D., Perez, E., Schiefer, N., Hatfield-Dodds, Z., DasSarma, N., Tran-Johnson, E., et al. Language models (mostly) know what they know. *arXiv preprint arXiv:2207.05221*, 2022.

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

Khattar, V. and Jin, M. Winning the citylearn challenge: adaptive optimization with evolutionary search under trajectory-based guidance. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 14286–14294, 2023.

Khattar, V., Ding, Y., Sel, B., Lavaei, J., and Jin, M. A cmdp-within-online framework for meta-safe reinforcement learning. In *The Eleventh International Conference on Learning Representations*, 2022.

Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., and Iwasawa, Y. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35: 22199–22213, 2022.

Lanham, T., Chen, A., Radhakrishnan, A., Steiner, B., Denison, C., Hernandez, D., Li, D., Durmus, E., Hubinger, E., Kernion, J., et al. Measuring faithfulness in chain-of-thought reasoning. *arXiv preprint arXiv:2307.13702*, 2023.

Liang, P., Bommasani, R., Lee, T., Tsipras, D., Soylu, D., Yasunaga, M., Zhang, Y., Narayanan, D., Wu, Y., Kumar, A., et al. Holistic evaluation of language models. *arXiv preprint arXiv:2211.09110*, 2022.

Libby, M. E., Weiss, J. S., Bancroft, S., and Ahearn, W. H. A comparison of most-to-least and least-to-most prompting on the acquisition of solitary play skills. *Behavior analysis in practice*, 1:37–43, 2008.

Lin, T.-W., Khattar, V., Huang, Y., Hong, J., Jia, R., Liu, C.-C., Sangiovanni-Vincentelli, A., and Jin, M. Causalprompt: Enhancing llms with weakly supervised causal reasoning for robust per-formance in non-language tasks.

Liu, Y., Han, T., Ma, S., Zhang, J., Yang, Y., Tian, J., He, H., Li, A., He, M., Liu, Z., et al. Summary of chatgpt/gpt-4 research and perspective towards the future of large language models. *arXiv preprint arXiv:2304.01852*, 2023.

Long, J. Large language model guided tree-of-thought. *arXiv preprint arXiv:2305.08291*, 2023.

Lyu, Q., Havaldar, S., Stein, A., Zhang, L., Rao, D., Wong, E., Apidianaki, M., and Callison-Burch, C. Faithful chain-of-thought reasoning. *arXiv preprint arXiv:2301.13379*, 2023.

Merrill, W. and Sabharwal, A. The expresssive power of transformers with chain of thought. *arXiv preprint arXiv:2310.07923*, 2023.

Mialon, G., Dessì, R., Lomeli, M., Nalmpantis, C., Pasunuru, R., Raileanu, R., Rozière, B., Schick, T., Dwivedi-Yu, J., Celikyilmaz, A., et al. Augmented language models: a survey. *arXiv preprint arXiv:2302.07842*, 2023.

Monsell, S. Task switching. *Trends in cognitive sciences*, 7 (3):134–140, 2003.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

Robinson, J. and Wingate, D. Leveraging Large Language Models for Multiple Choice Question Answering. September 2022. URL https://openreview.net/forum?id=yKbprarjc5B.

Schuurmans, D. Memory augmented large language models are computationally universal. *arXiv preprint arXiv:2301.04589*, 2023.

Sel, A., Sel, B., and Kasnakoglu, C. Glsdc based parameter estimation algorithm for a pmsm model. *Energies*, 14(3): 611, 2021.

Sel, A., Sel, B., Coskun, U., and Kasnakoglu, C. Sos-based nonlinear observer design for simultaneous state and disturbance estimation designed for a pmsm model. *Sustainability*, 14(17):10650, 2022.

Sel, B., Tawaha, A., Ding, Y., Jia, R., Ji, B., Lavaei, J., and Jin, M. Learning-to-learn to guide random search: Derivative-free meta blackbox optimization on manifold. In *Learning for Dynamics and Control Conference*, pp. 38–50. PMLR, 2023.

Sel, B., Shanmugasundaram, P., Kachuee, M., Zhou, K., Jia, R., and Jin, M. Skin-in-the-game: Decision making via multi-stakeholder alignment in llms. *arXiv preprint arXiv:2405.12933*, 2024.

Shao, Z., Gong, Y., Shen, Y., Huang, M., Duan, N., and Chen, W. Synthetic Prompting: Generating Chain-of-Thought Demonstrations for Large Language Models. June 2023. URL https://openreview.net/forum?id=RYD1UMgTdk.

Sloman, S. A. The empirical case for two systems of reasoning. *Psychological bulletin*, 119(1):3, 1996.

Srivastava, A., Rastogi, A., Rao, A., Shoeb, A. A. M., Abid, A., Fisch, A., Brown, A. R., Santoro, A., Gupta, A., Garriga-Alonso, A., et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*, 2022.

Suzgun, M., Scales, N., Schärli, N., Gehrmann, S., Tay, Y., Chung, H. W., Chowdhery, A., Le, Q. V., Chi, E. H., Zhou, D., and Wei, J. Challenging BIG-Bench Tasks and Whether Chain-of-Thought Can Solve Them, October 2022. URL http://arxiv.org/abs/2210.09261. arXiv:2210.09261 [cs].

Thoppilan, R., De Freitas, D., Hall, J., Shazeer, N., Kulshreshtha, A., Cheng, H.-T., Jin, A., Bos, T., Baker, L., Du, Y., et al. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*, 2022.

Turpin, M., Michael, J., Perez, E., and Bowman, S. R. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. *arXiv preprint arXiv:2305.04388*, 2023.

Wang, X., Wei, J., Schuurmans, D., Le, Q. V., Chi, E. H., Narang, S., Chowdhery, A., and Zhou, D. Self-Consistency Improves Chain of Thought Reasoning in Language Models. September 2022. URL https://openreview.net/forum?id=1PL1NIMMrw.

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., Chi, E. H., Hashimoto, T., Vinyals, O., Liang, P., Dean, J., and Fedus, W. Emergent Abilities of Large Language Models, October 2022a. URL http://arxiv.org/abs/2206.07682. arXiv:2206.07682 [cs].

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022b.

Wu, C.-J., Raghavendra, R., Gupta, U., Acun, B., Ardalani, N., Maeng, K., Chang, G., Aga, F., Huang, J., Bai, C., et al. Sustainable ai: Environmental implications, challenges and opportunities. *Proceedings of Machine Learning and Systems*, 4:795–813, 2022.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., and Narasimhan, K. Tree of Thoughts: Deliberate Problem Solving with Large Language Models, May 2023. URL http://arxiv.org/abs/2305.10601. arXiv:2305.10601 [cs].

Zelikman, E., Wu, Y., Mu, J., and Goodman, N. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

Zhang, Z., Zhang, A., Li, M., and Smola, A. Automatic Chain of Thought Prompting in Large Language Models. September 2022. URL https://openreview.net/forum?id=5NTt8GFjUHkr.

Zhou, D., Schärli, N., Hou, L., Wei, J., Scales, N., Wang, X., Schuurmans, D., Cui, C., Bousquet, O., Le, Q. V., and Chi, E. H. Least-to-Most Prompting Enables Complex Reasoning in Large Language Models. September 2022a. URL https://openreview.net/forum?id=WZH7099tgfM.

Zhou, H., Nova, A., Larochelle, H., Courville, A., Neyshabur, B., and Sedghi, H. Teaching algorithmic reasoning via in-context learning. *arXiv preprint arXiv:2211.09066*, 2022b.

A. Game of 24 - Additional Details

In order to avoid confusion in our analysis of AoT in the game of 24, we give additional details in terms of terminologies we use as well as their direct implications in the performance figures. An Illustration of these are given in Fig. 7.

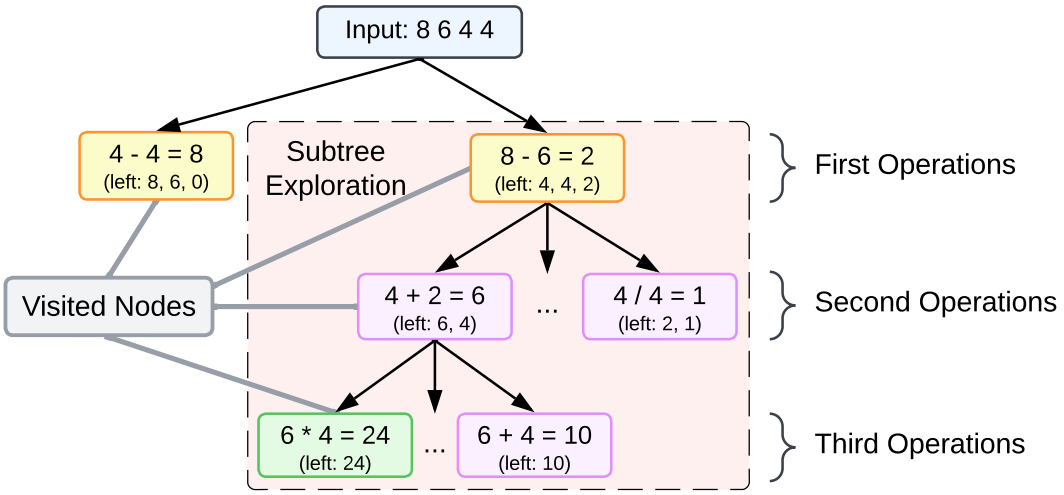

Figure 7. An illustration of terminologies we use for the game of 24. The yellow nodes represent the first operations and the states they lead to; the green node represents the node where we find the solution; all other nodes are represented by pink.

First operations / First iterations. This represents the scenario that after we choose the first two number in the game of 24, the case of either adding, subtracting, multiplying or dividing them.

Subtree Exploration. This denotes searching all or most of the nodes coming from the same state, typically states with less than four numbers left.

Number of nodes visited. This is the number of states that the method has been on the game of 24. Each state is the set of number we are left with, after our operations in the numbers. For example, after the first operation we might be left with the numbers '$8\ 3\ 1$'. This set of numbers represent a state, as well as the state of '$8\ 3$' that we will be left with after another operation of '$8 * 1 = 8$'.

B. Creative Writing

We use the creative writing task, also used by (Yao et al., 2023), where the LLM is provided with four arbitrary sentences. The objective is to craft a cohesive narrative divided into four paragraphs, with each paragraph culminating in one of the given sentences. This exercise not only fosters creativity but also emphasizes strategic deliberation.

B.1. Task Setup

Sentences are randomly sourced from *randomwordgenerator.com*, resulting in 100 distinct sets of inputs. Given the absence of predetermined correct answers, the primary focus lies in evaluating the coherence of the responses. We have noted that GPT-4 consistently aligns with these input guidelines. Evaluation is centered around assessing passage coherence using a GPT-4 zero-shot prompt, where each output is rated on a scale of 1 to 10. Each task response undergoes five such evaluations, with their scores being averaged subsequently.

B.2. Baselines

For this task, both standard and CoT prompts are employed without preliminary training. While the standard prompt directly guides the LLM to fashion a cohesive narrative based on stipulated parameters, the CoT prompt obliges the model to initially

outline a succinct plan prior to drafting the narrative, serving as an intermediate cognitive bridge. For each task iteration, ten samples are generated using both the standard and CoT methods. Results of the ToT approach are presented without modification.

B.3. AoT Setup

Mirroring ToT's methodology, the task is tackled in a zero-shot setting. Our prompt instructs the model to first formulate five distinct plans. Subsequent to this, the model selects the most promising among them to shape a narrative and then refines it for optimal coherence. The exact prompts used for this zero-shot approach will be provided in the subsequent section.

B.4. Results

As depicted in Fig. 8, AoT outpaces other singular query prompting techniques such as standard prompting and CoT in terms of performance. It also exhibits a marked improvement over ToT, although the difference is not statistically significant. Comprehensive scores, along with the average query count needed for each method, are consolidated in Table 9. Notably, AoT necessitates fewer queries compared to ToT.

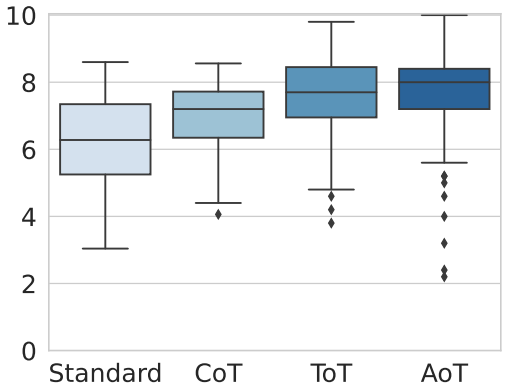

Figure 8. Comparison of the standard prompting, CoT, ToT and AoT on the creative writing task.

Method	Score	Avg. Queries
Standard Prompting	6.19	1
CoT	6.93	1
ToT	7.56	20
AoT	**7.58**	1

Table 9. Performance of the methods determined by GPT-4.

C. CoT vs. Single Iteration AoT in the Game of 24

To demonstrate that the tree search mechanism is fundamentally distinct from the CoT prompting, even in scenarios where AoT's in-context examples include only a single initial operation in the game of 24, we draw a comparison between AoT (Short) and CoT. In this setup, AoT (Short) determines the first operation and subsequently conducts a tree search on the remaining three numbers. Interestingly, AoT (Short) achieves a success rate of 48%, while CoT lags significantly, securing only 4%. These results underscore the notion that even a rudimentary search mechanism can lead to significant performance enhancements.

D. Detailed Analysis on the Effect of the Length of the Prompts

In this section, we delve deeper into Fig. 6 by presenting histograms for the successful, unsuccessful, and total games of '24', considering the number of initial steps in methods AoT (Short), AoT, and AoT (Long). These are displayed in Figs. 9-11.

From these figures, it becomes evident that the length of the prompts, measured by the number of initial steps included in in-context examples, correlates with the length of their solutions to test examples. This trend is consistent across all three cases, suggesting that AoT's strategy in determining the number of initial steps is influenced by its in-context examples.

Interestingly, when AoT is provided a well-balanced set of initial steps that emphasize the most promising operations, it excels in solving the majority of games in earlier iterations. This indicates AoT's capacity to prioritize swift problem-solving without sacrificing performance. This tendency is also observed in AoT (Long), albeit with a somewhat reduced success rate, as illustrated in Fig. 9.

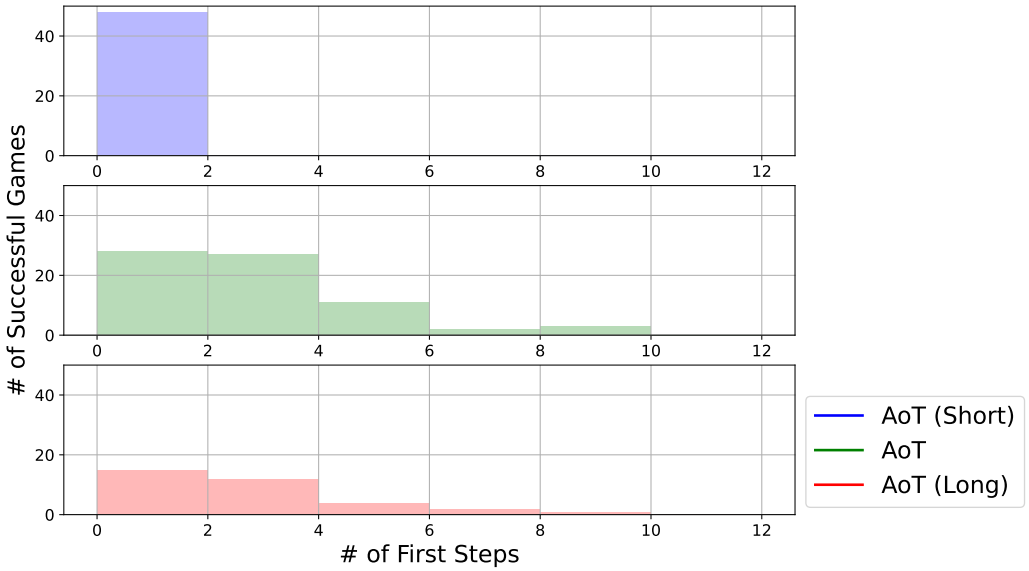

Figure 9. Histogram of the number of successful games with respect to the number of first steps for AoT (Short), AoT and AoT (Long).

E. Proof of Corollary 3.1

Corollary E.1. *Consider* $\text{TIME}(a^n)$ *as the class of languages L for which a Turing machine exists that operates within a time complexity of* $O(a^n)$ *for some* $a \geq 1$. *For a transformer generating* $O(a^n)$ *intermediate tokens with AoT, we have*

$$\text{TIME}(a^n) \subseteq \text{AoT}(n), \tag{2}$$

where $\text{AoT}(n)$ *refers to the decoding steps by AoT when the input has* n *tokens.*

Proof. Since with AoT prompting, we can backtrack and continue from other nodes, the number of intermediate tokens scale exponentially with the depth of the problem over the number of possible actions in each leaf node. Then, directly by Theorem 2. in Merrill & Sabharwal (2023), we have the stated result implying the capability of solving NP problems with AoT prompting. □

F. Prompts

F.1. Game of 24

Below, we represent the specific prompts employed for the various methods detailed in the experiments section. It's important to note that the terms "System","User", and "Assistant" are utilized to denote the *roles* within the OpenAI API

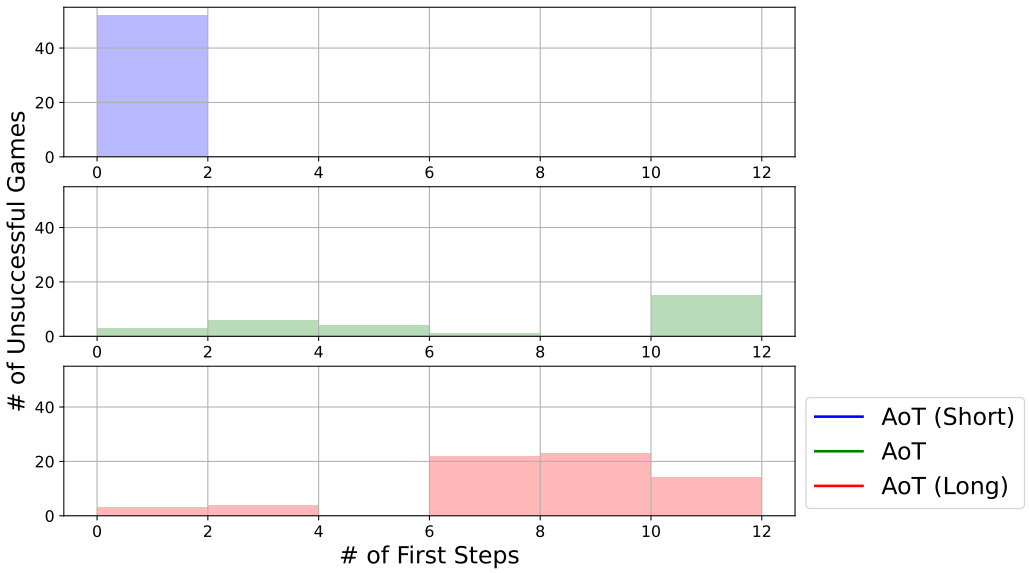

Figure 10. Histogram of the number of unsuccessful games with respect to the number of first steps for AoT (Short), AoT and AoT (Long).

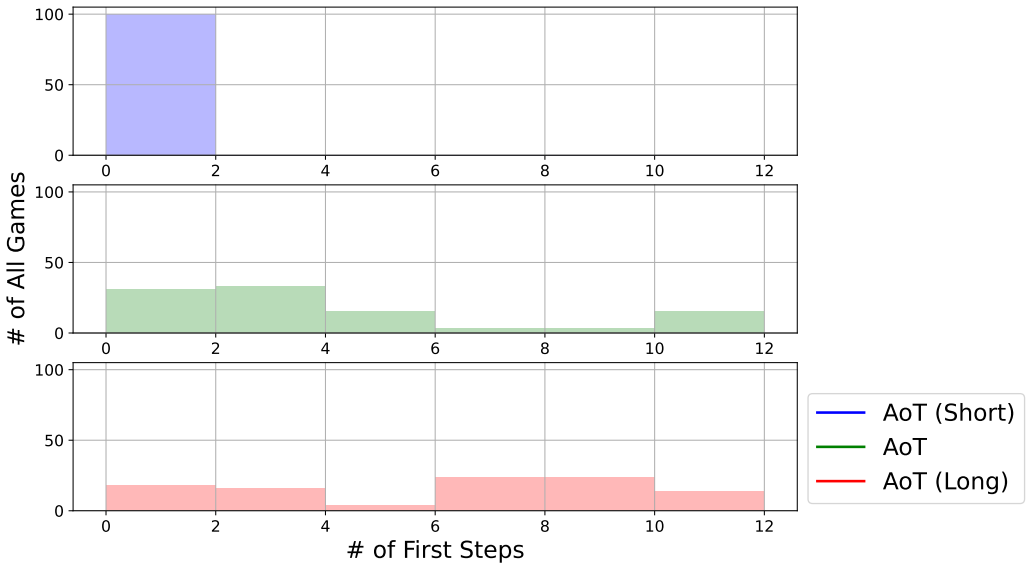

Figure 11. Histogram of the number of all games with respect to the number of first steps for AoT (Short), AoT and AoT (Long).

when operating in chat completion mode. The line breaks serve to show the transitions between the user and assistant interactions within the API.

AoT (DFS)

```
System:
        Use numbers and basic arithmetic operations (+ - * /) to obtain
            24. When considering the next steps, do not choose operations
            that will result in a negative or fractional number. In order
            to help with the calculations, the numbers in the parenthesis
            represent the numbers that are left after the operations and
```

```
            they are in descending order.

        Another thing we do is when there are only two numbers left in the
             parenthesis, we check whether we can arrive at 24 only by
            using basic arithmetic operations (+ - * /). Some examples
            regarding this idea:
        (21 2) no
        since 21 + 2 = 23, 21 - 2 = 19, 21 * 2 = 42, 21 / 2 = 10.5, none
            of which is equal to 24.
        (30 6) 30 - 6 = 24 yes
        (8 3) 8 * 3 = 24 yes
        (12 8) no
        (48 2) 48 / 2 = 24 yes

        Most importantly, do not give up, all the numbers that will be
            given has indeed a solution.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
User:
        14 8 8 2
Assistant:
        Trying a promising first operation:
        1. 8 + 2: (14, 10, 8)
        - 14 + 10: (24, 8) 32, 16, 192, 3
        - 14 - 10: (8, 4) 12, 8, 32, 2
        - 14 * 10: (140, 8) 148, 132, 1120, fractional
        - 14 / 10: (8, 1.4) fractional
        - 14 + 8: (22, 10) 32, 12, 220, fractional
        - 14 - 8: (10, 6) 16, 4, 60, fractional
        - 14 * 8: (112, 10) 122, 102, 1120, fractional
        - 14 / 8: fractional
        - 10 + 8: (18, 14) 32, 4, 252, fractional
        - 10 - 8: (14, 2) 16, 12, 28, 7
        - 10 * 8: (80, 14) 94, 66, big, fractional
        - 10 / 8: fractional

        Trying another promising first operation:
        2. 8 / 2: (14, 8, 4)
        - 14 + 8: (22, 4) 26, 18, 88, fractional
        - 14 - 8: (6, 4) 10, 2, 24 = 6 * 4 -> found it!
        Backtracking the solution:
        Step 1:
        8 / 2 = 4
        Step 2:
        14 - 8 = 6
        Step 3:
        6 * 4 = 24
        Considering these steps: 24 = 6 * 4 = (14 - 8) * 4 = (14 - 8) * (8
             / 2) = 24.
        answer: (14 - 8) * (8 / 2) = 24.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
User:
        9 5 5 5
Assistant:
        Trying a promising first operation:
```

```
        1. 9 - 5: (5, 5, 4)
        - 5 + 5: (10, 4) 14, 6, 40, fractional
        - 5 - 5: (4, 0) 4, 4, 0, undefined
        - 5 * 5: (25, 4) 29, 21, 100, fractional
        - 5 / 5: (4, 1) 5, 3, 4, 4
        - 5 + 4: (9, 5) 14, 4, 45, fractional
        - 5 - 4: (5, 1) 6, 4, 5, 0.2
        - 5 * 4: (20, 5) 25, 15, 100, fractional
        - 5 / 4: fractional

        Trying another promising first operation:
        2. 5 + 5: (10, 9, 5)
        - 10 + 9: (19, 5) 24 = 19 + 5 -> found it!
        Backtracking the solution:
        Step 1:
        5 + 5 = 10
        Step 2:
        10 + 9 = 19
        Step 3:
        19 + 5 = 24
        Considering these steps: 24 = 19 + 5 = (10 + 9) + 5 = ((5 + 5) +
           9) + 5 = 24.
        answer: ((5 + 5) + 9) + 5 = 24.
~~~
```

```
        - 9 + 9: (18, 10) 28, 8, 180, fractional
        - 9 - 9: (10, 0) 10, 10, 0, undefined
        - 9 * 9: (81, 10) 91, 71, 810, fractional
        - 9 / 9: (10, 1) 11, 9, 10, 10

        Trying another promising first operation:
        2. 10 - 4: (13, 9, 6)
        - 13 + 9: (22, 6) 28, 16, 132, fractional
        - 13 - 9: (6, 4) 10, 2, 24 = 6 * 4 -> found it!
        Backtracking the solution:
        Step 1:
        10 - 4 = 6
        Step 2:
        13 - 9 = 4
        Step 3:
        6 * 4 = 24
        Considering these steps: 24 = 6 * 4 = 6 * (13 - 9) = (10 - 4) *
           (13 - 9) = 24.
        answer: (10 - 4) * (13 - 9) = 24.
~~~
```

```
        5 - 2 = 3
        Step 3:
        8 * 3 = 24
        Considering these steps: 24 = 8 * 3 = 8 * (5 - 2) = 8 * (5 - (8 /
            4)) = 24.
        answer: 8 * (5 - (8 / 4)) = 24.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
User:
        11 11 1 1
Assistant:
        Trying a promising first operation:
        1. 11 - 1: (11, 10, 1)
        - 11 + 10: (21, 1) 22, 20, 21, 21
        - 11 - 10: (1, 1) 2, 0, 1, 1
        - 11 * 10: (110, 1) 111, 109, 110, 110
        - 11 / 10: fractional
        - 11 + 1: (12, 10) 22, 2, 120, 1.2
        - 11 - 1: (10, 10) 20, 0, 100, 1
        - 11 * 1: (11, 10) 21, 1, 110, 1.1
        - 11 / 1: (11, 10) 21, 1, 110, fractional
        - 10 + 1: (11, 11) 22, 0, 121, 1
        - 10 - 1: (11, 9) 20, 2, 99, 1.1
        - 10 * 1: (11, 10) 21, 1, 110, 1.1
        - 10 / 1: (11, 10) 21, 1, 110, 1.1

        Trying another promising first operation:
        2. 11 + 1: (12, 11, 1)
        - 12 + 11: (23, 1) 24 = 23 + 1 -> found it!
        Backtracking the solution:
        Step 1:
        11 + 1 = 12
        Step 2:
        12 + 11 = 23
        Step 3:
        23 + 1
        Considering these steps: 24 = 23 + 1 = (12 + 11) + 1 = ((11 + 1) +
            11) + 1 = 24.
        answer: ((11 + 1) + 11) + 1 = 24.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
User:
        11 7 4 1
Assistant:
        Trying a promising first operation:
        1. 7 * 4: (28, 11, 1)
        - 28 + 11: (39, 1) 40, 38, 39, 39
        - 28 - 11: (17, 1) 18, 16, 17, 17
        - 28 * 11: (308, 1) 309, 307, 308, 308
        - 28 / 11: fractional
        - 28 + 1: (29, 11) 40, 18, 319, fractional
        - 28 - 1: (27, 11) 38, 16, 297, fractional
        - 28 * 1: (28, 11) 39, 17, 308, fractional
        - 28 / 1: (28, 11) 39, 17, 308, fractional
        - 11 + 1: (29, 28) 57, 1, 812, fractional
        - 11 - 1: (28, 10) 38, 18, 280, fractional
```

```
        - 11 * 1: (28, 11) 39, 17, 308, fractional
        - 11 / 1: (28, 11) 39, 17, 308, fractional

        Trying another promising first operation:
        2. 7 + 1: (11 8 4)
        - 11 + 8: (19, 4) 23, 15, 76, fractional
        - 11 - 8: (4, 3) 7, 1, 12, fractional
        - 11 * 8: (88, 4) 92, 84, 352, fractional
        - 11 / 8: fractional
        - 11 + 4: (15, 8) 23, 7, 120, fractional
        - 11 - 4: (7, 8) 15, -1, 56, fractional
        - 11 * 4: (44, 8) 52, 36, 352, fractional
        - 11 / 4: fractional
        - 8 + 4: (12, 11) 23, -1, 132, fractional
        - 8 - 4: (11, 4) 15, 7, 44, fractional
        - 8 * 4: (32, 11) 43, 21, 352, fractional
        - 8 / 4: (11, 2) 13, 9, 22, fractional

        Trying another promising first operation:
        3. 4 + 1: (11 7 5)
        - 11 + 7: (18, 5) 23, 13, 90, fractional
        - 11 - 7: (5, 4) 9, 1, 20, fractional
        - 11 * 7: (77, 5) 82, 72, 385, fractional
        - 11 / 7: fractional
        - 11 + 5: (16, 7) 23, 9, 112, fractional
        - 11 - 5: (7, 6) 13, 1, 42, fractional
        - 11 * 5: (55, 7) 62, 48, 385, fractional
        - 11 / 5: fractional
        - 7 + 5: (12, 11) 23, 1, 132, fractional
        - 7 - 5: (11, 2) 13, 9, 22, fractional
        - 7 * 5: (35, 11) 46, 24 = 35 - 11 -> found it!
        Step 1:
        4 + 1 = 5
        Step 2:
        7 * 5 = 35
        Step 3:
        35 - 11 = 24
        Considering these steps: Backtracking the solution:
        24 = 35 - 11 = (7 * 5) - 11 = (7 * (4 + 1)) - 11 = 24.
        answer: (7 * (4 + 1)) - 11 = 24.
~~~
```

```
        - 11 + 3: (14, 20) 34, -6, 280, fractional
        - 11 - 3: (8, 20) 28, -12, 160, fractional
        - 11 * 3: (33, 20) 53, 13, 660, fractional
        - 11 / 3: fractional

        Trying another promising first operation:
        2. 11 - 3: (8, 5, 4)
        - 8 + 5: (13, 4) 17, 9, 52, fractional
        - 8 - 5: (4, 3) 7, 1, 12, fractional
        - 8 * 5: (40, 4) 44, 36, 160, fractional
        - 8 / 5: fractional
        - 8 + 4: (12, 5) 17, 7, 60, fractional
        - 8 - 4: (5, 4) 9, 1, 20, fractional
        - 8 * 4: (32, 5) 37, 27, 160, fractional
        - 8 / 4: (5, 2) 7, 3, 10, 2.5
        - 5 + 4: (9, 8) 17, 1, 72, fractional
        - 5 - 4: (8, 1) 9, 7, 8, 8
        - 5 * 4: (20, 8) 28, 12, 160, fractional
        - 5 / 4: fractional

        Trying another promising first operation:
        3. 11 * 3: (33, 5, 4)
        - 33 + 5: (38, 4) 42, 34, 152, fractional
        - 33 - 5: (28, 4) 32, 24 = 28 - 4 -> found it!
        Backtracking the solution:
        Step 1:
        11 * 3 = 33
        Step 2:
        33 - 5 = 28
        Step 3:
        28 - 4 = 24
        Considering these steps: 24 = 28 - 4 = (33 - 5) - 4 = ((11 * 3) -
           5) - 4 = 24.
        answer: ((11 * 3) - 5) - 4 = 24.
~~~
```

```
        2. 13 + 5: (18, 12, 2)
        - 18 + 12: (30, 2) 32, 28, 60, fractional
        - 18 - 12: (6, 2) 8, 4, 12, 3
        - 18 * 12: (216, 2) 218, 214, 432, fractional
        - 18 / 12: fractional
        - 18 + 2: (20, 12) 32, 8, 240, fractional
        - 18 - 2: (16, 12) 28, 4, 192, fractional
        - 18 * 2: (36, 12) 48, 24 = 36 - 12 -> found it!
        Backtracking the solution:
        Step 1:
        13 + 5 = 18
        Step 2:
        18 * 2 = 36
        Step 3:
        36 - 12 = 24
        Considering these steps: 24 = 36 - 12 = (18 * 2) - 12 = ((13 + 5)
            * 2) - 12 = 24.
        answer: ((13 + 5) * 2) - 12 = 24.
~~~
```

```
        Another thing we do is when there are only two numbers left in the
            parenthesis, we check whether we can arrive at 24 only by
           using basic arithmetic operations (+ - * /). Some examples
           regarding this idea:
        (21 2) no
        since 21 + 2 = 23, 21 - 2 = 19, 21 * 2 = 42, 21 / 2 = 10.5, none
           of which is equal to 24.
        (30 6) 30 - 6 = 24 yes
        (8 3) 8 * 3 = 24 yes
        (12 8) no
        (48 2) 48 / 2 = 24 yes

        Most importantly, do not give up, all the numbers that will be
           given has indeed a solution.
User:
        14 8 8 2
Assistant:
        Trying a promising first operation:
        1. 8 + 2: (14, 10, 8)
        - 14 + 10: (24, 8) 32, 16, 192, 3
        - 14 - 10: (8, 4) 12, 8, 32, 2
        - 14 * 10: (140, 8) 148, 132, 1120, fractional
        - 14 / 10: (8, 1.4) fractional
        - 14 + 8: (22, 10) 32, 12, 220, fractional
        - 14 - 8: (10, 6) 16, 4, 60, fractional
        - 14 * 8: (112, 10) 122, 102, 1120, fractional
        - 14 / 8: fractional
        - 10 + 8: (18, 14) 32, 4, 252, fractional
        - 10 - 8: (14, 2) 16, 12, 28, 7
        - 10 * 8: (80, 14) 94, 66, big, fractional
        - 10 / 8: fractional

        Trying another promising first operation:
        2. 14 + 8: (22, 8, 2)
        - 22 + 8: (30, 2) 32, 28, 60, 15
        - 22 - 8: (14, 2) 16, 12, 28, 7
        - 22 * 8: (176, 2) 178, 174, 88
        - 22 / 8: (2.75, 2) fractional
        - 22 + 2: (24, 8) 32, 16, 192, 3
        - 22 - 2: (20, 8) 28, 12, 160, fractional
        - 22 * 2: (44, 8) 52, 36, 352, fractional
        - 22 / 2: (11, 8) 19, 3, 88, fractional
        - 8 + 2: (22, 10) 32, 12, 220, fractional
        - 8 - 2: (22, 6) 28, 16, 132, fractional
        - 8 * 2: (22, 16) 38, 6, 352, fractional
        - 8 / 2: (22, 4) 26, 18, 88, fractional

        Trying another promising first operation:
        3. 14 + 2: (16, 8, 8)
        - 16 + 8: (24, 8) 32, 16, 192, 3
        - 16 - 8: (8, 8) 16, 0, 64, 1
        - 16 * 8: (128, 8) 136, 120, 1024, 16
        - 16 / 8: (8, 2) 10, 6, 16, 4
```

```
- 8 + 8: (16, 16 32, 0, 256, 1
- 8 - 8: (16, 0) 16, 16, 0, undefined
- 8 * 8: (64, 16) 80, 48, 1024, 4
- 8 / 8: (16, 1) 17, 15, 16, 16

Trying another promising first operation:
4. 8 - 2: (14, 8, 6)
- 14 + 8: (22, 14) 36, 8, 308, fractional
- 14 - 8: (6, 6) 12, 0, 36, 1
- 14 * 8: (112, 6) 118, 106, 672, fractional
- 14 / 8: (6, 1.75) fractional
- 14 + 6: (20, 8) 22, 12, 160, fractional
- 14 - 6: (8, 8) 16, 0, 64, 1
- 14 * 6: (84, 8) 92, 76, 672, fractional
- 14 / 6: (8, 2.3) fractional
- 8 + 6: (14, 14) 28, 0, 196, 1
- 8 - 6: (14, 2) 16, 12, 28, 7
- 8 * 6: (48, 14) 62, 34, 672, fractional
- 8 / 6: (14, 1.3) fractional

Trying another promising first operation:
5. 8 * 2: (16, 14, 8)
- 16 + 14: (30, 8) 38, 22, 240, fractional
- 16 - 14: (8, 2) 10, 6, 16, 4
- 16 * 14: (224, 8) 232, 216, 1792, 28
- 16 / 14: (8, 1.1) fractional
- 16 + 8: (24, 14) 38, 10, 336, fractional
- 16 - 8: (14, 8) 22, 6, 112, fractional
- 16 * 8: (128, 14) 142, 112, 1792, fractional
- 16 / 8: (14, 2) 16, 12, 28, 7
- 14 + 8: (22, 16) 38, 6, 352, fractional
- 14 - 8: (16, 6) 22, 10, 96, fractional
- 14 * 8: (112, 16) 128, 96, 1792, 7
- 14 / 8: (16, 1.7) fractional

Trying another promising first operation:
6. 14 * 2: (28, 8, 8)
- 28 + 8: (36, 8) 44, 28, 288, fractional
- 28 - 8: (20, 8) 28, 12, 160, fractional
- 28 * 8: (224, 8) 232, 216, 1792, 28
- 28 / 8: (8, 3.5) fractional, fractional, 28, fractional
- 8 + 8: (16, 16 32, 0, 256, 1
- 8 - 8: (16, 0) 16, 16, 0, undefined
- 8 * 8: (64, 16) 80, 48, 1024, 4
- 8 / 8: (16, 1) 17, 15, 16, 16

Trying another promising first operation:
7. 8 / 2: (14, 8, 4)
- 14 + 8: (22, 4) 26, 18, 88, fractional
- 14 - 8: (6, 4) 10, 2, 24 = 6 * 4 -> found it!
Backtracking the solution:
Step 1:
8 / 2 = 4
Step 2:
```

```
        14 - 8 = 6
        Step 3:
        6 * 4 = 24
        Considering these steps: 24 = 6 * 4 = (14 - 8) * 4 = (14 - 8) * (8
            / 2) = 24.
        answer: (14 - 8) * (8 / 2) = 24.
~~~
```

```
        - 9 - 5: (25, 4) 29, 21, 100, fractional
        - 9 * 5: (45, 25) 70, 20, 1125, fractional
        - 9 / 5: (25, 1.8) fractional, fractional, 45, fractional
        - 9 + 1: (10, 5) 15, 5, 50, 2
        - 9 - 1: (8, 5) 13, 3, 40, fractional
        - 9 * 1: (9, 5) 14, 4, 45, fractional
        - 9 / 1: (9, 5) 14, 4, 45, fractional
        - 5 + 1: (9, 6) 15, 3, 54, fractional
        - 5 - 1: (9, 4) 13, 5, 36, fractional
        - 5 * 1: (9, 5) 14, 4, 45, fractional
        - 5 / 1: (9, 5) 14, 4, 45, fractional

        Trying another promising first operation:
        5. 9 * 5: (45, 5, 5)
        - 45 + 5: (50, 5) 55, 45, 250, 10
        - 45 - 5: (40, 5) 45, 35, 200, 8
        - 45 * 5: (225, 5) 230, 220, 1125, 45
        - 45 / 5: (9, 5) 14, 4, 45, fractional
        - 5 + 5: (10, 4) 14, 6, 40, fractional
        - 5 - 5: (4, 0) 4, 4, 0, undefined
        - 5 * 5: (25, 4) 29, 21, 100, fractional
        - 5 / 5: (4, 1) 5, 3, 4, 4

        Trying another promising first operation:
        6. 5 + 5: (10, 9, 5)
        - 10 + 9: (19, 5) 24 = 19 + 5 -> found it!
        Backtracking the solution:
        Step 1:
        5 + 5 = 10
        Step 2:
        10 + 9 = 19
        Step 3:
        19 + 5 = 24
        Considering these steps: 24 = 19 + 5 = (10 + 9) + 5 = ((5 + 5) +
           9) + 5 = 24.
        answer: ((5 + 5) + 9) + 5 = 24.
~~~
```

```
        - 8 - 6: (2, 0) 2, 2, 0, undefined
        - 8 * 6: (48, 0) 48, 48, 0, undefined
        - 8 / 6: (1.3, 0) fractional
        - 8 + 0: (8, 6) 14, 2, 48, fractional
        - 8 - 0: (8, 6) 14, 2, 48, fractional
        - 8 * 0: (6, 0) 6, 6, 0, undefined
        - 8 / 0: undefined
        - 6 + 0: (8, 6) 14, 2, 48, fractional
        - 6 - 0: (8, 6) 14, 2, 48, fractional
        - 6 * 0: (8, 0) 8, 8, 0, undefined
        - 6 / 0: undefined

        Trying another promising first operation:
        3. 4 / 4: (8, 6, 1)
        - 8 + 6: (14, 1) 15, 13, 14, 14
        - 8 - 6: (2, 1) 3, 1, 2, 2
        - 8 * 6: (48, 1) 49, 47, 48, 48
        - 8 / 6: (1.3, 1) fractional
        - 8 + 1: (9, 6) 15, 3, 54, fractional
        - 8 - 1: (7, 6) 13, 1, 42, fractional
        - 8 * 1: (8, 6) 14, 2, 48, fractional
        - 8 / 1: (8, 6) 14, 2, 48, fractional
        - 6 + 1: (8, 7) 15, 1, 56, fractional
        - 6 - 1: (8, 5) 13, 3, 40, fractional
        - 6 * 1: (8, 6) 14, 2, 48, fractional
        - 6 / 1: (8, 1) 9, 7, 8, 8

        Trying another promising first operation:
        4. 8 - 6: (4, 4, 2)
        - 4 + 4: (8, 2) 10, 6, 16, 4
        - 4 - 4: (2, 0) 2, 2, 0, undefined
        - 4 * 4: (16, 2) 18, 14, 32, 8
        - 4 / 4: (2, 1) 3, 1, 2, 2
        - 4 + 2: (6, 4) 10, 2, 24 = 6 * 4 -> found it!
        Backtracking the solution:
        Step 1:
        8 - 6 = 2
        Step 2:
        4 + 2 = 6
        Step 3:
        6 * 4 = 24
        Considering these steps: 24 = 6 * 4 = (4 + 2) * 4 = (4 + (8 - 6))
           * 4 = 24.
        answer: (4 + (8 - 6)) * 4 = 24.
~~~
```

```
- 9 + 9: (18, 10) 28, 8, 180, fractional
- 9 - 9: (10, 0) 10, 10, 0, undefined
- 9 * 9: (81, 10) 91, 71, 810, fractional
- 9 / 9: (10, 1) 11, 9, 10, 10
Trying another promising first operation:
2. 13 / 10: (9, 4, 1.3)
- 9 + 4: (13, 1.3) fractional, fractional, fractional, 10
- 9 - 4: (5, 1.3) fractional
- 9 * 4: (36, 1.3) fractional
- 9 / 4: (2.3, 1.3) fractional, 1, fractional, fractional
- 9 + 1.3: (10.3, 4) fractional
- 9 - 1.3: (7.7, 4) fractional
- 9 * 1.3: (11.7, 4) fractional
- 9 / 1.3: (6.9, 4) fractional
- 4 + 1.3: (9, 5.3) fractional
- 4 - 1.3: (9, 2.7) fractional
- 4 * 1.3: (9, 5.2) fractional
- 4 / 1.3: (9, 3.1) fractional
Trying another promising first operation:
3. 9 / 4: (13, 10, 2.3)
- 13 + 10: (23, 2.3) fractional, fractional, fractional, 10
- 13 - 10: (3, 2.3) fractional
- 13 * 10: (130, 2.3) fractional
- 13 / 10: (2.3, 1.3) fractional, 1, fractional, fractional
- 13 + 2.3: (15.3, 10) fractional, fractional, 153, fractional
- 13 - 2.3: (11.7, 10) fractional, fractional, 117, fractional
- 13 * 2.3: (29.9, 10) fractional, fractional, 299, fractional
- 13 / 2.3: (10, 5.6) fractional, fractional, 560, fractional
- 10 + 2.3: (13, 12.3) fractional
- 10 - 2.3: (13, 7.7) fractional
- 10 * 2.3: (23, 13) 36, 10, 299, fractional
- 10 / 2.3: (13, 4.3) fractional
Trying another promising first operation:
4. 13 / 4: (10, 9, 3.3)
- 10 + 9: (19, 3.3) fractional
- 10 - 9: (3.3, 1) fractional
- 10 * 9: (90, 3.3) fractional
- 10 / 9: (3.3, 1.1) fractional, fractional, fractional, 3
- 10 + 3.3: (13.3, 9) fractional
- 10 - 3.3: (9, 6.7) fractional
- 10 * 3.3: (33, 9) 42, 24, 297, fractional
- 10 / 3.3: (3.1, 9) fractional
- 9 + 3.3: (12.3, 10) fractional, fractional, 123, fractional
- 9 - 3.3: (10, 5.7) fractional, fractional, 57, fractional
- 9 * 3.3: (29.7, 10) fractional, fractional, 297, fractional
- 9 / 3.3: (10, 2.7) fractional, fractional, 27, fractional
Trying another promising first operation:
5. 13 / 9: (10, 9, 1.4)
- 10 + 9: (19, 1.4) fractional
- 10 - 9: (1.4, 1) fractional
- 10 * 9: (90, 1.4) fractional, fractional, 130, fractional
- 10 / 9: (1.1, 1.4) fractional
- 10 + 1.4: (11.4, 9) fractional
- 10 - 1.4: (9, 8.6) fractional
```

```
        - 10 * 1.4: (140, 9) 149, 131, 1260, fractional
        - 10 / 1.4: (9, 7.1) fractional
        - 9 + 1.4: (10.4, 10) fractional, fractional, 104, fractional
        - 9 - 1.4: (10, 7.6) fractional, fractional, 76, fractional
        - 9 * 1.4: (12.6, 10) fractional, fractional, 126, fractional
        - 9 / 1.4: (10, 6.4) fractional, fractional, 64, fractional
        Trying another promising first operation:
        6. 10 - 4: (13, 9, 6)
        - 13 + 9: (22, 6) 28, 16, 132, fractional
        - 13 - 9: (6, 4) 10, 2, 24 = 6 * 4 -> found it!
        Backtracking the solution:
        Step 1:
        10 - 4 = 6
        Step 2:
        13 - 9 = 4
        Step 3:
        6 * 4 = 24
        Considering these steps: 24 = 6 * 4 = 6 * (13 - 9) = (10 - 4) *
           (13 - 9) = 24.
        answer: (10 - 4) * (13 - 9) = 24.
~~~
```

```
        (12 * 2) * (10 - 9) = 24.
        answer: (12 * 2) * (10 - 9) = 24.
~~~
```

```
            numbers, we will be able to get to 24.

        So, when we reach 24 in the third steps, that means we found the
            solution, we directly backtrack to write the answer.

        In each step, we want to choose the best operations so that the
            numbers left, will have a good chance to reach 24.
User:
        14 8 8 2
Assistant:
        Let's consider the most promising possible first steps:
        1. 8 + 8: (16, 14, 2)
        2. 14 - 8: (8, 6, 2)
        3. 14 + 2: (16, 8, 8)
        4. 8 / 2: (14, 8, 4)

        Let's consider the most promising second steps:
        1. (16, 14, 2)
        - 16 - 14: (2, 2)
        - 16 / 2: (8, 14)
        - 14 * 2: (28, 16)
        - 14 / 2: (7, 16)

        2. (8, 6, 2)
        - 8 - 6: (2, 2)
        - 8 / 2: (4, 6)
        - 6 * 2: (12, 8)
        - 6 / 2: (3, 8)

        3. (16, 8, 8)
        - 16 - 8: (8, 8)
        - 16 / 8: (2, 8)
        - 8 * 8: (64, 16)
        - 8 / 8: (1, 16)

        4. (14, 8, 4)
        - 14 - 8: (6, 4)
        - 14 / 4: (3.5, 8)
        - 8 * 4: (32, 14)
        - 8 / 4: (2, 14)

        Let's consider the most promising third steps:
        1. 8 + 8 = 16
        - 16 - 14: (2, 2): 4, 0, 4, 1
        - 16 / 2: (8, 14): 22, 6, 112, fractional
        - 14 * 2: (28, 16): 44, 12, 448, fractional
        - 14 / 2: (7, 16): 23, 9, 112, fractional

        2. 14 - 8 = 6
        - 8 - 6: (2, 2): 4, 0, 4, 1
        - 8 / 2: (6, 4): 10, 2, 24 = 6 * 4 -> found it!
        Backtracking the solution:
        Step 1:
        14 - 8 = 6
```

```
        Step 2:
        8 / 2 = 4
        Step 3:
        6 * 4 = 24
        Considering these steps: 24 = 6 * 4 = (14 - 8) * (8 / 2) = 24.
        answer: (14 - 8) * (8 / 2) = 24.
~~~
```

```
        10 + 9 = 19
        Step 3:
        19 + 5 = 24
        Considering these steps: 24 = 19 + 5 = (10 + 9) + 5 = ((5 + 5) +
            9) + 5 = 24.
        answer: ((5 + 5) + 9) + 5 = 24.
~~~
```

```
        3. 8 - 6 = 2
        - 4 * 4: (16, 2): 19, 14, 32, 8
        - 4 + 2: (6, 4): 10, 2, 24 = 6 * 4 -> found it!
        Backtracking the solution:
        Step 1:
        8 - 6 = 2
        Step 2:
        4 + 2 = 6
        Step 3:
        6 * 4 = 24
        Considering these steps: 24 = 6 * 4 = (4 + 2) * 4 = (4 + (8 - 6))
            * 4 = 24.
        answer: (4 + (8 - 6)) * 4 = 24.
~~~
```

```
        2. 10 - 4 = 6
        - 9 + 6: (15, 13): 28, 2, 195, fractional
        - 9 * 6: (54, 13): 67, 41, 702, fractional
        - 13 - 9: (6, 4): 10, 2, 24 = 6 * 4 -> found it!
        Backtracking the solution:
        Step 1:
        13 - 9 = 4
        Step 2:
        10 - 4 = 6
        Step 3:
        6 * 4 = 24
        Considering these steps: 24 = 6 * 4 = (10 - 4) * (13 - 9) = 24.
        answer:  (10 - 4) * (13 - 9) = 24.
~~~
```

```
        - 8 / 8: (9, 1): 10, 8, 9, 9

        2. 8 / 4 = 2
        - 5 - 2: (8, 3): 11, 5, 24 = 8 * 3 -> found it!
        Backtracking the solution:
        Step 1:
        8 / 4 = 2
        Step 2:
        5 - 2 = 3
        Step 3:
        8 * 3 = 24
        Considering these steps: 24 = 8 * 3 = 8 * (5 - 2) = 8 * (5 - (8 /
            4)) = 24.
        answer: 8 * (5 - (8 / 4)) = 24.
~~~
```

```
        - 11 / 1: (11, 10): 21, 1, 110, 0

        2. 11 + 1 = 12
        - 12 + 11: (23, 1): 24 = 23 + 1 -> found it!
        Backtracking the solution:
        Step 1:
        11 + 1 = 12
        Step 2:
        12 + 11 = 23
        Step 3:
        23 + 1 = 24
        Considering these steps: 24 = 23 + 1 = (12 + 11) + 1 = ((11 + 1) +
            11) + 1 = 24.
        answer: ((11 + 1) + 11) + 1 = 24.
~~~
```

```
        - 11 - 1: (28, 10): 38, 18, 280, 2.8

        2. 7 + 1 = 8
        - 11 + 8: (19, 4): 23, 15, 76, 4.75
        - 8 + 4: (12, 11): 23, 7, 132, 3
        - 11 - 8: (4, 3): 7, 1, 12, 1.33
        - 8 - 4: (7, 11): 18, 4, 77, 1.75

        3. 4 + 1 = 5
        - 11 - 5: (7, 6): 13, 1, 42, 1.17
        - 7 - 5: (11, 2): 13, 9, 22, 5.5
        - 7 * 5: (35, 11): 46, 24 = 35 - 11 -> found it!
        Backtracking the solution:
        Step 1:
        4 + 1 = 5
        Step 2:
        7 * 5 = 35
        Step 3:
        35 - 11 = 24
        Considering these steps: 24 = 35 - 11 = (7 * 5) - 11 = (7 * (4 +
           1)) - 11 = 24.
        answer: (7 * (4 + 1)) - 11 = 24.
        ~~~
```

```
        - 7 + 5: (12, 11)
        - 7 - 5: (2, 11)

        Let's consider the most promising third steps to reach 24:
        1. 11 * 3 = 33
        - 33 - 5: (28, 4): 32, 24 = 28 - 4 -> found it!
        Backtracking the solution:
        Step 1:
        11 * 3 = 33
        Step 2:
        33 - 5 = 28
        Step 3:
        28 - 4 = 24
        Considering these steps: 24 = 28 - 4 = (33 - 5) - 4 = ((11 * 3) -
           5) - 4 = 24.
        answer: ((11 * 3) - 5) - 4 = 24.
~~~
```

```
        Backtracking the solution:
        Step 1:
        12 / 2 = 6
        Step 2:
        13 + 6 = 19
        Step 3:
        19 + 5 = 24
        Considering these steps: 24 = 19 + 5 = (13 + 6) + 5 = (13 + (12 /
            2)) + 5 = 24.
        answer: (13 + (12 / 2)) + 5 = 24.
        ~~~
```

```
        Step 3:
        16 + 8 = 24
        Considering these steps: 24 = 16 + 8 = (8 * 2) + 8 = ((9 - 1) * 2)
            + 8 = 24.
        answer: ((9 - 1) * 2) + 8 = 24.
~~~
```

```
        Trying a promising first operation:
        1. 5 + 5: (10, 9, 5)
        - 10 + 9: (19, 5) 24 = 19 + 5 -> found it!
        Backtracking the solution:
        Step 1:
        5 + 5 = 10
        Step 2:
        10 + 9 = 19
        Step 3:
        19 + 5 = 24
        Considering these steps: 24 = 19 + 5 = (10 + 9) + 5 = ((5 + 5) +
           9) + 5 = 24.
        answer: ((5 + 5) + 9) + 5 = 24.
~~~
```

```
        8 8 5 4
Assistant:
        Trying a promising first operation:
        1. 8 / 4: (8, 5, 2)
        - 8 + 5: (13, 2) 15, 11, 26, fractional
        - 8 - 5: (3, 2) 5, 1, 6, 1.5
        - 8 * 5: (40, 2) 42, 38, 80, 20
        - 8 / 5: fractional
        - 8 + 2: (10, 5) 15, 5, 50, 2
        - 8 - 2: (6, 5) 11, 1, 30, fractional
        - 8 * 2: (16, 5) 21, 11, 80, fractional
        - 8 / 2: (5, 4) 9, 1, 20, fractional
        - 5 + 2: (7, 8) 15, 1, 56, fractional
        - 5 - 2: (8, 3) 11, 5, 24 = 8 * 3 -> found it!
        Backtracking the solution:
        Step 1:
        8 / 4 = 2
        Step 2:
        5 - 2 = 3
        Step 3:
        8 * 3 = 24
        Considering these steps: 24 = 8 * 3 = 8 * (5 - 2) = 8 * (5 - (8 /
            4)) = 24.
        answer: 8 * (5 - (8 / 4)) = 24.
~~~
```

```
            - 11 / 5: fractional
            - 7 + 5: (12, 11) 23, 1, 132, fractional
            - 7 - 5: (11, 2) 13, 9, 22, fractional
            - 7 * 5: (35, 11) 46, 24 = 35 - 11 -> found it!
            Step 1:
            4 + 1 = 5
            Step 2:
            7 * 5 = 35
            Step 3:
            35 - 11 = 24
            Considering these steps: Backtracking the solution:
            24 = 35 - 11 = (7 * 5) - 11 = (7 * (4 + 1)) - 11 = 24.
            answer: (7 * (4 + 1)) - 11 = 24.
~~~
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
User:
        9 8 2 1
Assistant:
        Trying a promising first operation:
        1. 9 - 1: (8, 8, 2)
        - 8 + 8: (16, 2) 18, 14, 32, 8
        - 8 - 8: (2, 0) 2, 2, 0, undefined
        - 8 * 8: (64, 2) 66, 62, 128, 32
        - 8 / 8: (2, 1) 3, 1, 2, 2
        - 8 + 2: (10, 8) 18, 2, 80, 4
        - 8 - 2: (6, 8) 14, -2, 48, fractional
        - 8 * 2: (16, 8) 24 = 16 + 8 -> found it!
        Backtracking the solution:
        Step 1:
        9 - 1 = 8
        Step 2:
        8 * 2 = 16
        Step 3:
        16 + 8 = 24
        Considering these steps: 24 = 16 + 8 = (8 * 2) + 8 = ((9 - 1) * 2)
            + 8 = 24.
        answer: ((9 - 1) * 2) + 8 = 24.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

F.3. 5×5 Mini Crosswords Prompts

In the following we give all the exact prompts we used for the methods given in the experiments section and some selection of the generations from the LLM. In the prompts the keywords "System", "User" and "Assistant" used are the *roles* in the OpenAI API in chat completion mode. The line breaks are to show the turns between the user and assistant in the API.

F.3.1. AoT

```
User:
        h1. A lunar valley
        h2. A fatty oil
        h3. To entice
        h4. To lower; to reduce
        h5. A solitary person
        v1. According to the roster
        v2. Another name for Port-Francqui
        v3. An illicit lover; a European lake
        v4. To lisp
        v5. To come in

        The words I already found are:
        h1. rille
        h3. tempt
        v2. ilebo
        v5. enter
Assistant:
        There are two horizontal and two vertical words. Since they are
            the amount, let's consider the next unfound horizontal word,
```

```
    which is h2. Since we are considering the second horizontal
    word, we need to look at the second letters of the vertical
    words we found:
- What is the second letter of ilebo: e
- What is the second letter of enter: n
The word we are looking fits the clue "A fatty oil" and needs to
    have the letters e and n in its (v2, two) second and (v5, five)
     fifth letters: _ e _ _ n. The candidates are: oleic, olein,
    olive.
- oleic (o l e i c, _ e _ _ n) doesn't fit
- olein (o l e i n, _ e _ _ n) fits
- olive (o l i v e, _ e _ _ n) doesn't fit
We add the word olein for h2. The current words are:
h1. rille
h2. olein
h3. tempt
v2. ilebo
v5. enter
Now, let's consider the next unfound vertical word, which is v1.
    Since we are considering the first vertical word, we need to
    look at the first letters of the horizontal words we found:
- What is the first letter of rille: r
- What is the first letter of olein: o
- What is the first letter of tempt: t
The word we are looking fits the clue "According to the roster"
    and needs to have the letters r, o, and t in its (h1, one)
    first, (h2, two) second, and (h3, three) third letters: r o t _
     _. The candidates are: none
Now, let's consider the next unfound vertical word, which is v3.
    Since we are considering the third vertical word, we need to
    look at the third letters of the horizontal words we found:
- What is the third letter of rille: l
- What is the third letter of olein: e
- What is the third letter of tempt: m
The word we are looking fits the clue "An illicit lover; a
    European lake" and needs to have the letters l, e, and m in its
     (h1, one) first, (h2, two) second, and (h3, three) third
    letters: l e m _ _. The candidates are: leman.
- leman (l e m a n, l e m _ _) fits
We add the word leman for v3. The current words are:
h1. rille
h2. olein
h3. tempt
v2. ilebo
v3. leman
v5. enter
Now, let's consider the next unfound horizontal word, which is h4.
     Since we are considering the fourth horizontal word, we need
    to look at the fourth letters of the vertical words we found:
- What is the fourth letter of ilebo: b
- What is the fourth letter of leman: a
- What is the fourth letter of enter: e
The word we are looking fits the clue "To lower; to reduce" and
    needs to have the letters b, a, and e in its (v2, two) second,
```

```
            (v3, three) third, and (v5, five) fifth letters: _ b a _ e. The
             candidates are: abase, abate.
        - abase (a b a s e, _ b a _ e) fits
        - abate (a b a t e, _ b a _ e) fits
        Since both fits, let's consider another word.
        Now, let's consider the last unfound vertical word, which is v4.
            Since we are considering the fourth vertical word, we need to
            look at the fourth letters of the horizontal words we found:
        - What is the fourth letter of rille: l
        - What is the fourth letter of olein: i
        - What is the fourth letter of tempt: p
        The word we are looking fits the clue "To lisp" and needs to have
            the letters l, i, p, and s in its (h1, one) first, (h2, two)
            second, (h3, three) third letters: l i p _ _. The candidates
            are: The candidate is "lisps".
        - lisps (l i s p s, l i p _ _) doesn't fit

        We have considered all of the words for the horizontal words, we
            have two options: abase and abate, let's choose one of them.
        answer:
        h1. rille
        h2. olein
        h3. tempt
        h4. abase
        v2. ilebo
        v3. leman
        v5. enter
~~~
```

```
    to have the letters u and d in its (v1, one) first and (v4,
    four) fourth letters: u _ _ d _. The candidates are: uredo.
- uredo (u r e d o, u _ _ d _) fits
We add the word uredo for h2. The current words are:
h1. sawer
h2. uredo
h3. rater
v1. surge
v4. edema
Now, let's consider the next unfound vertical word, which is v2.
    Since we are considering the second vertical word, we need to
    look at the second letters of the horizontal words we found:
- What is the second letter of sawer: a
- What is the second letter of uredo: r
- What is the second letter of rater: a
The word we are looking fits the clue "The Brazilian macaw; an
    Australian bird" and needs to have the letters a, r, and a in
    its (h1, one) first, (h2, two) second, and (h3, third) third
    letters: a r a _ _. The candidates are: arara.
- arara (a r a r a, a r a _ _) fits
We add the word arara for v2. The current words are:
h1. sawer
h2. uredo
h3. rater
v1. surge
v2. arara
v4. edema
Now, let's consider the next unfound horizontal word, which is h4.
     Since we are considering the fourth horizontal word, we need
    to look at the fourth letters of the vertical words we found:
- What is the fourth letter of surge: g
- What is the fourth letter of arara: r
- What is the fourth letter of edema: m
the word we are looking fits the clue "Pasture land" and needs to
    have the letters g, r, and m in its (v1, one) first, (v2, two)
    second, and (v4, four) fourth letters: g r _ m _. The
    candidates are: grama.
- grama (g r a m a, g r a _ _) fits
We add the word grama for h4. The current words are:
h1. Sawer
h2. Uredos
h3. Rater
h4. Grama
v1. surge
v2. arara
v4. edema
Now, let's consider the next unfound vertical word, which is v3.
    Since we are considering the third vertical word, we need to
    look at the third letters of the horizontal words we found:
- What is the third letter of sawer: w
- What is the third letter of uredo: e
- What is the third letter of rater: t
- What is the third letter of grama: a
The word we are looking fits the clue "A Timorese island" and
```

```
            needs to have the letters w, e, t, and a in its (h1, one) first
            , (h2, two) second, (h3, three) third, and (h4, four) fourth
            letters: w e t a _. The candidates are: The candidate is "wetar
            ".
        - wetar (w e t a r, w e t a _) fits
        We add the word wetar for v3. The current words are:
        h1. sawer
        h2. uredo
        h3. rater
        h4. grama
        v1. surge
        v2. arara
        v3. wetar
        v4. edema
        Now, let's consider the last unfound horizontal word, which is h5.
             Since we are considering the fifth horizontal word, we need to
             look at the fifth letters of the vertical words we found:
        - What is the fifth letter of surge: e
        - What is the fifth letter of arara: a
        - What is the fifth letter of wetar: r
        - What is the fifth letter of edema: a
        The word we are looking fits the clue "Receiving by the ear" and
            needs to have the letters e, a, r, and a in its (v1, one) first
            , (v2, two) second, (v3, three) third, and (v4, four) fourth
            letters: e a r a _. The candidates are: The candidate is "earal
            ".
        - earal (e a r a l, e a r a _) fits
        We add the word earal for h5. The current words are:
        h1. sawer
        h2. uredo
        h3. rater
        h4. grama
        h5. earal
        v1. surge
        v2. arara
        v3. wetar
        v4. edema
        Since we found all horizontal words:
        answer:
        h1. sawer
        h2. uredo
        h3. rater
        h4. grama
        h5. earal
        v1. surge
        v2. arara
        v3. wetar
        v4. edema
~~~
```

```
        h5. A bishop's permission for a priest to leave a diocese
        v1. To steal; to brush across
        v2. A sedge (a primitive three-sided grass)
        v3. Grape jam
        v4. A flatworm larva
        v5. Ore refuse; to prepare material for glass by heat

        h2. waver
        h5. exeat
        v1. swipe
        v2. carex
Assistant:
        Let's consider the next unfound horizontal word, which is h1.
            Since we are considering the first horizontal word, we need to
            look at the first letters of the vertical words we found:
        - What is the first letter of swipe: s
        - What is the first letter of carex: c
        The word we are looking fits the clue "Dandruff; scum; the bull-
            trout" and needs to have the letters s and c in its (v1, one)
            first and (v2, two) second letters: s c _ _ _. The candidates
            are: scurf.
        - scurf (s c u r f, s c _ _ _) fits
        We add the word scurf for h1. The current words are:
        h1. scurf
        h2. waver
        h5. exeat
        v1. swipe
        v2. carex
        Now, let's consider the next unfound vertical word, which is v3.
            Since we are considering the third vertical word, we need to
            look at the third letters of the horizontal words we found:
        - What is the third letter of scurf: u
        - What is the third letter of waver: v
        - What is the third letter of exeat: e
        The word we are looking fits the clue "Grape jam" and needs to
            have the letters u, v, and e in its (h1, one) first, (h2, two)
            second, and (h5, five) fifth letters: u v _ _ e. The candidates
             are: The candidate is "uvetta".
        - uvetta (u v e t t a, u v _ _ e) doesn't fit
        Let's consider the next unfound vertical word, which is v4. Since
            we are considering the fourth vertical word, we need to look at
             the fourth letters of the horizontal words we found:
        - What is the fourth letter of scurf: r
        - What is the fourth letter of waver: e
        - What is the fourth letter of exeat: a
        The word we are looking fits the clue "A flatworm larva" and needs
             to have the letters r, e, and a in its (h1, one) first, (h2,
            two) second, and (h5, five) fifth letters: r e _ _ a. The
            candidates are: redia.
        - redia (r e d i a, r e _ _ a) fits
        We add the word redia for v4. The current words are:
        h1. scurf
        h2. waver
        h5. exeat
```

```
v1. swipe
v2. carex
v4. redia
Now, let's consider the next unfound horizontal word, which is h3.
     Since we are considering the third horizontal word, we need to
     look at the third letters of the vertical words we found:
- What is the third letter of swipe: i
- What is the third letter of carex: r
- What is the third letter of redia: d
The word we are looking fits the clue "A Turkish written decree"
    and needs to have the letters i, r, and d in its (v1, one)
    first, (v2, two) second, and (v4, four) fourth letters: i r _ d
     _. The candidates are: irade.
- irade (i r a d e, i r _ d _) fits
We add the word irade for h3. The current words are:
h1. scurf
h2. waver
h3. irade
h5. exeat
v1. swipe
v2. carex
v4. redia
Now, let's consider the last unfound vertical word, which is v5.
    Since we are considering the fifth vertical word, we need to
    look at the fifth letters of the horizontal words we found:
- What is the fifth letter of scurf: f
- What is the fifth letter of waver: r
- What is the fifth letter of irade: e
- What is the fifth letter of exeat: t
The word we are looking fits the clue "Ore refuse; to prepare
    material for glass by heat" and needs to have the letters f, r,
     e, and t in its (h1, one) first, (h2, two) second, (h3, three)
     third, and (h5, five) fifth letters: f r e _ t. The candidates
     are: The candidate is "frett".
- frett (f r e t t, f r e _ t) fits
We add the word frett for v5. The current words are:
h1. scurf
h2. waver
h3. irade
h5. exeat
v1. swipe
v2. carex
v4. redia
v5. frett
Since we found all horizontal words:
answer:
h1. scurf
h2. waver
h3. irade
h5. exeat
v1. swipe
v2. carex
v4. redia
v5. frett
```

```
~~~
```

```
Plan 4:
Your plan here.

Plan 5:
Your plan here.

Secondly, given an instruction and several plans, decide which choice
    is most promising. Analyze each choice in detail, then conclude in
    the last line "The best choice is {{s}}", where s the integer id of
     the choice.

Thirdly, write the passage according to that chosen plan in the most
    coherent way. Add "Passage:" before writing the passage under it.

Passage:
Your passage here.

Finally, refine the passage in the most coherent way, but you still
    have to end each paragraph with the given sentences as before.

Final Passage:
Final passage here.
```

F.4.2. SCORE PROMPT

```
Analyze the following passage, then at the last line conclude "Thus
    the coherency score is {{s}}", where s is an integer from 1 to 10.

{0}
```