Reasoning Abilities of Large Language Models: In-Depth Analysis on the Abstraction and Reasoning Corpus

SEUNGPIL LEE*, Electrical Engineering and Computer Science, GIST, Republic of Korea
WOOCHANG SIM*, AI Graduate School, GIST, Republic of Korea
DONGHYEON SHIN*, AI Graduate School, GIST, Republic of Korea
WONGYU SEO, Electrical Engineering and Computer Science, GIST, Republic of Korea
JIWON PARK, AI Graduate School, GIST, Republic of Korea
SEOKKI LEE, AI Graduate School, GIST, Republic of Korea
SANHA HWANG, AI Graduate School, GIST, Republic of Korea
SEJIN KIM, AI Graduate School, GIST, Republic of Korea
SUNDONG KIM, AI Graduate School, GIST, Republic of Korea

The existing methods for evaluating the inference abilities of Large Language Models (LLMs) have been predominantly results-centric, making it challenging to assess the inference process comprehensively. We introduce a novel approach using the Abstraction and Reasoning Corpus (ARC) benchmark to evaluate the inference and contextual understanding abilities of LLMs in a process-centric manner, focusing on three key components from the Language of Thought Hypothesis (LoTH): Logical Coherence, Compositionality, and Productivity. Our carefully designed experiments reveal that while LLMs demonstrate some inference capabilities, they still significantly lag behind human-level reasoning in these three aspects. The main contribution of this paper lies in introducing the LoTH perspective, which provides a method for evaluating the reasoning process that conventional results-oriented approaches fail to capture, thereby offering new insights into the development of human-level reasoning in artificial intelligence systems.

Contents

*The first three authors contributed equally and should be considered co-first authors.

1 Introduction

Recent Large Language Models (LLMs) have demonstrated performance levels close to that of humans, but experimental results showed that they lacked planning ability through thought or reasoning [6]. Consequently, a key question in recent language model research is: Can LLMs think? To address this question, new benchmarks for measuring reasoning abilities, such as MathVista [37], Bongard-Logo [44], and Raven [75], have been proposed. Among these, the Abstraction and Reasoning Corpus (ARC) [8] emerged to be one of the representative benchmarks for assessing reasoning abilities. As shown in Fig. 1 below, each task in ARC consists of 2–5 demonstration example pairs and a test example input grid. The goal is to infer rules from the given demonstration example pairs and apply them to the test example. Input and output grid sizes can vary from a minimum of 1×1 to a maximum of 30×30, with each grid having up to 10 different colors.

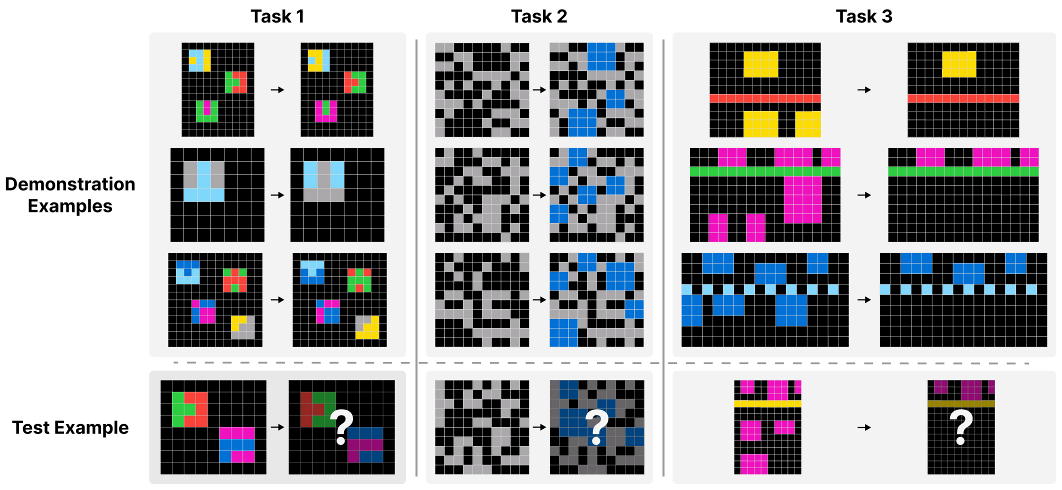


Fig. 1. Three different ARC tasks. Each task involves demonstration examples of input and output grids that exemplify the required transformation. Solvers must generate the correct output grid for the test example's input grid by applying the same transformation. ARC is a straightforward benchmark that can be solved using only four types of prior knowledge: objectness, goal-directedness, arithmetic, and geometric topology. Despite the small amount of prior knowledge required to solve the tasks, it presents a high level of reasoning difficulty. These characteristics enable ARC to serve as a benchmark that fairly measures reasoning abilities.

The ARC remains an unsolved challenge despite its seemingly simple content and evaluation methods. It demands a high level of abstraction and multiple reasoning steps, which explains why conventional deep learning techniques have not achieved success. The best-performing models to date have only achieved an accuracy of 40-55% [30], while LLMs (GPT-4, PaLM) have shown an accuracy of around 10-20% [42]. Compared to the average human accuracy of 80% [27], these results suggest significant differences in reasoning and abstraction capabilities between humans and LLMs. However, in-depth research into how LLMs reason and how their reasoning differs from human reasoning is lacking. This gap has led to calls for a shift from a results-focused evaluation to a more nuanced analysis of the process [2, 7, 24, 72], indicating a need for a new perspective that evaluates reasoning abilities based on the process rather than just the outcome.

To overcome the limitations of result-oriented analysis in artificial intelligence, this study adopts an existing theory on what constitutes human reasoning ability. According to the Language of Thought Hypothesis (LoTH) [17], human reasoning encompasses three essential characteristics:

Logical Coherence, the ability to maintain consistency in reasoning; **Compositionality**, the capability to construct complex ideas from simpler components; and **Productivity**, the capacity to formulate an indefinite number of thoughts or solutions using a finite set of elements.

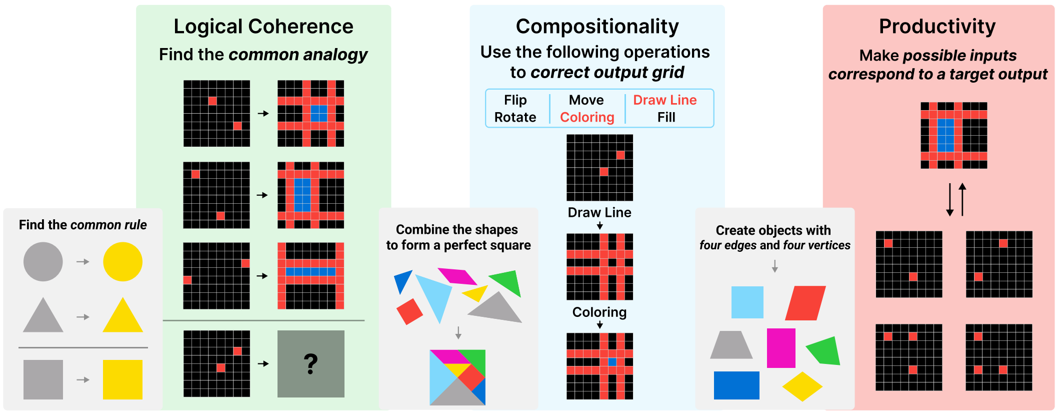

Logical Coherence
Find the *common analogy*
?
Find the *common rule*
Compositionality
Use the following operations to *correct output grid*
Flip
Rotate
Move
Coloring
Draw Line
Fill
Draw Line
Coloring
Combine the shapes to form a perfect square
Productivity
Make *possible inputs correspond to a target output*
Create objects with *four edges* and *four vertices*

Fig. 2. Three concepts of the Language of Thought Hypothesis (LoTH).

While attempts to evaluate logical coherence, compositionality, and productivity have existed before [6, 58], there were limitations in that the definitions of each component varied across papers and existing benchmarks showed insufficient performance in assessing each aspect. This study differs from previous research in two key ways: 1) by redefining concepts borrowed from psychology to fit into the field of computer science, and 2) by evaluating all elements through the visual reasoning benchmark ARC. To achieve these goals, we have designed three experiments:

(1) **Logical Coherence:** LoTH identifies two types of coherence. These are Inferential Coherence – the ability to apply logical reasoning across related instances coherently – and Semantic Coherence — the ability to maintain logical coherence in the reasoning process and results [18]. To verify both types of logical coherence, we augmented each solved ARC task with 100 similar test examples and evaluated the LLM's performance on these related instances. Additionally, we analyzed the solution processes, identifying cases where correct answers were derived from flawed reasoning, to measure the LLM's semantic coherence.

(2) **Compositionality:** Compositionality refers to a system's capacity to express one proposition being inherently linked to its ability to express related propositions [18]. In this study, we define compositionality as the ability to combine given semantics. Therefore, to evaluate compositionality, it is necessary to verify whether semantics can be combined as desired. Consequently, this study provided LLMs with step-by-step functions and examined whether they could identify the appropriate functions to solve ARC problems. Subsequently, we conducted an additional analysis to determine if the LLM could accurately predict the results from the given step-by-step functions and to understand the reasons for the failure.

(3) **Productivity:** Productivity refers to the ability to infinitely create unseen expressions by combining a limited set of semantics [18]. However, it is difficult to quantitatively measure whether one can make an infinite number of unseen expressions. Therefore, previous studies have evaluated productivity by assessing whether rule-compliant unseen expressions can be created [25, 31, 59]. Similarly, in this study, to evaluate the ability to generate unseen expressions, we examined whether unseen ARC tasks that comply with the rules could be generated when given a set of functions.

As a result, we have confirmed that the current level of LLMs possesses a basic understanding of images and is capable of simple types of compositional object manipulations. However, compared to human reasoning abilities, LLMs lag in three areas: 1) They are not inferentially and semantically coherent; 2) Their logical reasoning abilities, especially in a step-by-step manner, are weak; 3) They struggle with understanding and generating unseen representations under complex constraints.

Finally, this study summarizes and presents recent trends proposed to address the weaknesses in abstraction abilities and reasoning capabilities. Analyzing the reasoning abilities of LLMs according to the components of human reasoning and discussing how to enhance each component represents a differentiated approach from previous research. It offers a fresh perspective for measuring and advancing the reasoning capabilities of LLMs in the future.

2 Preliminaries

This section aims to explain why we chose the LoTH perspective and ARC before starting a detailed evaluation of LLMs' reasoning capabilities. First, we will look at existing definitions of reasoning abilities and show why LoTH is useful in the perspective of measuring intelligence in Section 2.1. Then, in Section 2.2, we show that the ARC is an appropriate benchmark for studying LLMs from the perspective of human reasoning, as it 1) utilizes abstract semantics that can be generalized, and 2) is easy to modify.

2.1 Limitation on Assessing Reasoning Ability of LLMs

Efforts to evaluate LLMs' capabilities continue, highlighting their strengths in image and text generation. Especially, analyses confirm that LLMs possess elements of a World Model [22], indicating potential in inference tasks. However, challenges in reasoning persist [58], with errors such as distortion and incomplete reasoning being frequently observed [34]. Studies indicate that complex compositionality remains a significant challenge [16].

The divergent claims about LLMs' reasoning abilities stem from result-centric measurement methods. Turing first shifted the approach toward a consequential direction [56], followed by others who focused on performance measurement [40, 49, 67]. Recently, Chollet attempted to quantify inference abilities from a consequential perspective [8]. However, these studies focus on what reasoning can achieve without specifying its constituent elements. West et al. [66] raised concerns about evaluating LLMs' reasoning abilities solely from this perspective.

To address these limitations, we propose adopting LoTH perspective. LoTH enhances discussions by integrating reasoning components with quantitative metrics, positing that inference involves the manipulation of mental representations with compositional syntax and combinatorial semantics. Our study evaluates LLMs' inference capabilities through LoTH, focusing on logical coherence, compositionality, and productivity.

Previous research has evaluated these aspects independently. Logical coherence refers to the ability to construct coherent logic in problem-solving [76]. Compositionality involves understanding and combining complex expressions [31]. Productivity is assessed by the accuracy and efficiency of output generated from limited resources [25, 59]. However, these attempts lack unified criteria and fail to provide a direct comparison to human reasoning processes.

Adopting the LoTH perspective offers strong justification for improving reasoning capabilities. It helps develop the ability to process information and solve tasks in a manner similar to human reasoning. Logical coherence ensures reasoning without contradictions, compositionality allows the adaption of known knowledge to new scenarios, and productivity enhances the capacity to generate results based on given rules. Thus, this approach aids LLMs in achieving more human-like reasoning, enabling them to address complex problems with innovative and valid results.

2.2 Advantages of using ARC as Reasoning Benchmark

The Abstraction and Reasoning Corpus (ARC) emerges as a compelling candidate for evaluating inference abilities from the perspective of LoTH. ARC aligns with LoTH by requiring the combinations of semantics to solve problems and allowing flexible task modifications.

2.2.1 Core Properties of ARC. ARC's key characteristic is its requirement to extract and combine compositional semantics, necessitating sophisticated problem-solving approaches. Two research findings support this:

(1) **Importance of Semantic Information:** Studies show that supplementary semantic information significantly improves ARC task performance. For instance, integrating graph-represented object information nearly doubled the success rate [71].
(2) **High Abstraction Level of ARC:** ARC's abstraction level surpasses that of other benchmarks [41]. Chollet argues that conventional feature extraction methods are insufficient for ARC, given its demand for complex shape interpretation and transformation comprehension [8].

These observations highlight the need to develop approaches that can effectively extract and utilize complex, abstract information for solving ARC tasks. Such properties fit with the perspective of LoTH, which views reasoning ability as arising from a combination of semantics.

2.2.2 Flexibility in benchmark adaptation. Despite its simple rules, ARC remains challenging, with LLMs achieving 15% accuracy [47], traditional program synthesis models reaching 26% [68], and the human average accuracy at 80% [27]. Various ARC variants have emerged to address this challenge:

(1) **1D-ARC** [71]: Reduces dimensionality from 2D to 1D, simplifying complexity while retaining core knowledge. It effectively addresses object cohesion challenges, achieving high LLM accuracy (approximately 90%).
(2) **MC-LARC** [51]: Adopts a multiple-choice format, transitioning from generative tasks to selection tasks. GPT-4 demonstrated strong performance (approximately 75%).
(3) **Mini-ARC** [28]: Limits grid size to 5x5, simplifying input while retaining 2D generative characteristics. Performance remains challenging, similar to the original ARC (approximately 15%).
(4) **ConceptARC** [43]: Organizes tasks into concept groups that focus on specific spatial and semantic concepts. Performance remains challenging, similar to the original ARC (approximately 20%).

These variations demonstrate ARC's transformation flexibility and emphasize the necessity of composition in solving ARC tasks. MC-LARC and 1D-ARC reduced reasoning step complexity, while Mini-ARC focused on reducing image complexity. The performance differences among these variants imply that reducing the need for complex transformation combinations can significantly improve results, highlighting the importance of combinatorial syntax in solving ARC.

In summary, the ARC emerges as a compelling benchmark for evaluating inference abilities through the lens of the LoTH. ARC's core strength lies in its requirement to extract and combine compositional semantics to solve tasks, as evidenced by improved performance with additional semantic information. The various ARC variants demonstrate flexibility for different experimental purposes, with their performance differences highlighting the necessity of combinatorial syntax in solving ARC tasks. Furthermore, ARC's high level of abstraction and reasoning complexity, shown by the significant gap between human and AI performance, validates its use as an effective tool for exploring inference abilities in the context of the LoTH.

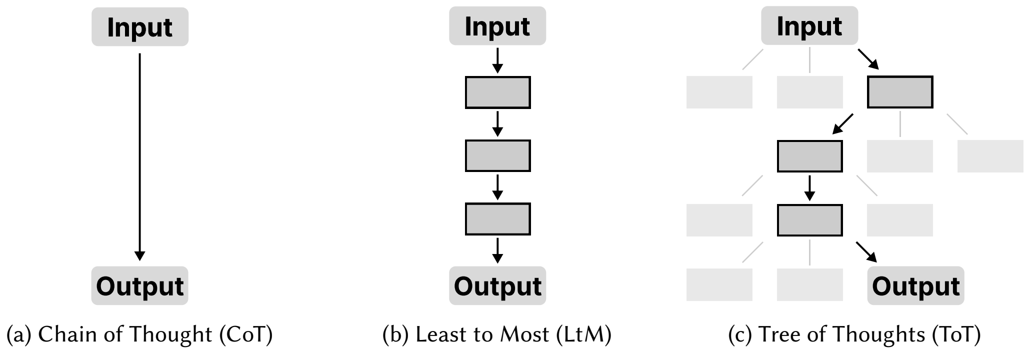
3 Evaluating the Inferential Capabilities of LLMs Using the ARC Benchmark

To evaluate whether LLMs possess inferential capabilities, one could compare these capabilities to human reasoning. As explained in Section 2.1, according to LoTH, human reasoning can be broadly categorized into three main components: Logical Coherence (Section 3.1), Compositionality (Section 3.2), and Productivity (Section 3.3). We utilized ARC to examine each aspect of the LLMs' reasoning capabilities from the perspective of LoTH.

3.1 Capability of LLMs 1: Logical Coherence

3.1.1 Motivation. Section 3.1 aims to evaluate the logical coherence of LLMs. This is a fundamental aspect of LoTH, which considers coherence in two dimensions: inferential coherence and semantic coherence [18]. **Semantic Coherence** refers to the ability to maintain logical coherence in the process and results of reasoning. **Inferential Coherence**, on the other hand, is a system's ability to consistently apply a specific type of logical inference across all relevant instances, given it can perform that inference in some cases. These concepts are crucial in human cognitive processes and relevant to the rule inference required in ARC tasks.

Our initial experiments primarily focused on measuring semantic coherence by evaluating whether the results produced by LLMs logically followed their problem-solving steps. This evaluation was conducted using various prompt techniques such as Chain of Thought (CoT) [65], Least to Most (LtM) [78], and Tree of Thought (ToT) [74], similar to previous ARC solving attempts [42, 71]. We compared the coherence levels these different prompting strategies achieved, aiming to identify which techniques yielded the most semantically coherent results across diverse problem-solving scenarios. However, recognizing the limitations of this approach in addressing inferential coherence, we introduced supplementary experiments using augmented ARC tasks. These tasks, created through the Re-ARC program [23], allowed us to assess how consistently LLMs can apply logical patterns across variations of originally solved problems, providing a more comprehensive evaluation of their logical reasoning capabilities.



(a) Chain of Thought (CoT) (b) Least to Most (LtM) (c) Tree of Thoughts (ToT)

Fig. 3. Three prompting techniques in the experiment about logical coherence: (a) CoT, (b) LtM, and (c) ToT.

3.1.2 Comparison Across Prompting Techniques. The perceived deficiency in LLMs' logical reasoning has been a recurrent critique, with direct attempts to solve ARC tasks yielding success rates below 10% [42]. To address this issue, enhancements in LLMs' logical reasoning are being pursued through prompting techniques such as CoT, LtM, and ToT. These strategies have been shown to effectively leverage LLMs' reasoning capabilities [61] and offer the advantage of providing a more transparent analysis for humans, as they involve a step-by-step reasoning process. Therefore, in this experiment, we assess the impact of these prompting strategies on LLMs' logical coherence by solving ARC tasks.

We applied three major prompting techniques – CoT, LtM, and ToT – to solve 100 ARC evaluation tasks using the GPT-4-32k model. Each technique was tested across five iterations. ARC tasks follow a few-shot learning paradigm, requiring the model to infer task rules from given example pairs and apply them to test examples. The CoT method enhances reasoning performance by generating answers through a structured chain of thought, which systematically connects steps required for solving ARC tasks and provides examples in the prompt. Similar contextual information was provided for LtM and ToT. LtM decomposes tasks into manageable steps and executes them sequentially, while ToT generates multiple candidates at each step post-decomposition, selecting the best candidate through a voting mechanism before proceeding to the next step.

Table 1. Averaged performance of each prompting technique. The accuracy is based on solving 100 random ARC tasks with CoT, LtM, and ToT prompts, each repeated five times. The accuracy outside the parentheses refers to the accuracy when only the results are correct, while the accuracy inside the parentheses indicates the accuracy when both the results and the process are correct.

Iteration	**CoT**	**LtM**	**ToT**
1	11% (3%)	6% (4%)	7% (3%)
2	10% (2%)	7% (4%)	4% (1%)
3	10% (5%)	6% (3%)	7% (2%)
4	10% (4%)	4% (2%)	7% (4%)
5	10% (6%)	5% (2%)	6% (2%)
Average	10.2% (4.0%)	5.6% (3.0%)	6.2% (2.4%)

Comparing ARC accuracy across prompts, CoT outperformed LtM and ToT in accuracy. Table 1 presents the results of applying LtM, CoT, and ToT to 100 randomly selected tasks from the ARC evaluation set. The experiment was repeated five times, with the percentage of correct answers included for each iteration. CoT achieved approximately 10% accuracy, while LtM and ToT showed about 6% accuracy. CoT demonstrates superior performance, while ToT and LtM suffer from cumulative error propagation, where small mistakes in one step of their multi-step answer generation process can lead to compounded errors in subsequent steps. Given CoT's accuracy (~11%) compared to LtM and ToT (~7%) and its resilience to error propagation, we exclusively used the CoT prompt in subsequent experiments.

However, when we checked the correctness of the solution process, all three prompting techniques showed low accuracy, with no significant difference at around 3%, as indicated in parentheses. These results demonstrate that while accuracy may differ depending on the prompting technique, there is little variation in semantic coherence. This consistency across prompting methods suggests that the issue lies not in the method of eliciting responses but in the fundamental reasoning capabilities of LLMs. It is also important to note that both the results and processes fall far short of the average human accuracy of 80%. These low performance metrics, particularly when compared to human benchmarks, cannot be attributed to the limitations of specific prompting techniques. These findings suggest that LLMs lag behind humans in terms of logical coherence. To analyze the specific reasons for this, we conducted follow-up experiments. Section 3.1.3 analyzes inferential coherence, one aspect of logical coherence, while Section 3.1.4 examines the semantic coherence of LLMs through case studies.

Sample Task

If the input grids are:

```
[[0, 3, 0, 0, 0, 0],
 [0, 3, 0, 2, 0, 0],
 [0, 0, 0, 2, 0, 0],
 [0, 8, 0, 0, 2, 2],
 [0, 0, 0, 0, 2, 2],
 [6, 6, 6, 0, 0, 0]]
```

then the correct output grids are:

```
[[0, 0, 0, 0, 3, 0],
 [0, 0, 0, 0, 3, 2],
 [0, 0, 0, 0, 0, 2],
 [0, 0, 0, 8, 2, 2],
 [0, 0, 0, 0, 2, 2],
 [0, 0, 0, 6, 6, 6]]
```

{additional examples}

To solve this task, follow the sub-tasks below.
1. Identify objects in the input grid.
2. Try to move each object to the right.
3. Stop when objects touch the right corner or other objects.

Following these steps will lead to the output grid.

Decomposing

If the input grids are:

```
[[0, 0, 0, 0, 0, 0],
 [0, 0, 3, 0, 0, 0],
 [0, 3, 0, 3, 0, 0],
 [0, 0, 3, 0, 3, 0],
 [0, 0, 0, 3, 0, 0],
 [0, 0, 0, 0, 0, 0]]
```

then the correct output grids are:

```
[[0, 0, 0, 0, 0, 0],
 [0, 0, 3, 0, 0, 0],
 [0, 3, 4, 3, 0, 0],
 [0, 0, 3, 4, 3, 0],
 [0, 0, 0, 3, 0, 0],
 [0, 0, 0, 0, 0, 0]]
```

{additional examples}

To solve this task, decompose the task into sub-tasks like below.
1. Identify the places surrounded by "3"s in the input grid.
2. Fill in the places you found with "4".

Following these steps will lead to the output grid.

Target Task

If the input grids are:

```
[[0, 0, 0, 0, 0, 0, 0],
 [0, 5, 8, 5, 0, 0, 0],
 [0, 5, 8, 5, 0, 0, 0],
 [0, 8, 8, 8, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0]]
```

then the correct output grids are:

```
[[0, 0, 0, 0, 0, 0, 0],
 [0, 8, 5, 8, 0, 0, 0],
 [0, 8, 5, 8, 0, 0, 0],
 [0, 5, 5, 5, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0]]
```

{additional examples}

If the input grid are:

```
[[0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 3, 2, 2, 0, 0, 0, 0, 0],
 [0, 3, 3, 2, 0, 0, 0, 0, 0],
 [0, 3, 2, 2, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 6, 6, 6, 0],
 [0, 0, 0, 0, 0, 1, 1, 1, 0],
 [0, 0, 0, 0, 0, 1, 6, 6, 0]]
```

What is the correct output grid?

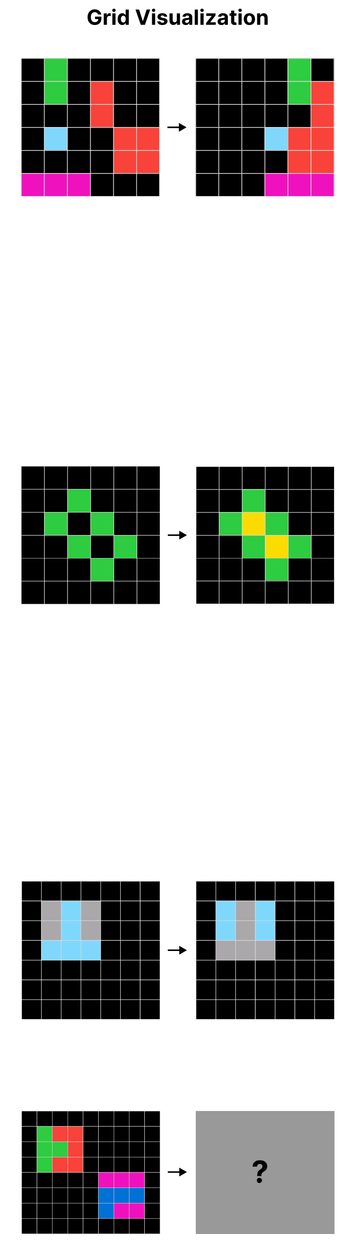


Fig. 4. Three types of prompts are shown on the left. Although all prompts are described as a 2D array of grids, we visualized them on the right for clarity. By default, all three techniques use prompts with two main components: a sample task and a target task. However, LtM and ToT use a different combination of the target task and its decomposition command. This difference arises because CoT strictly follows the given sub-task, while LtM and ToT decompose the task on their own.

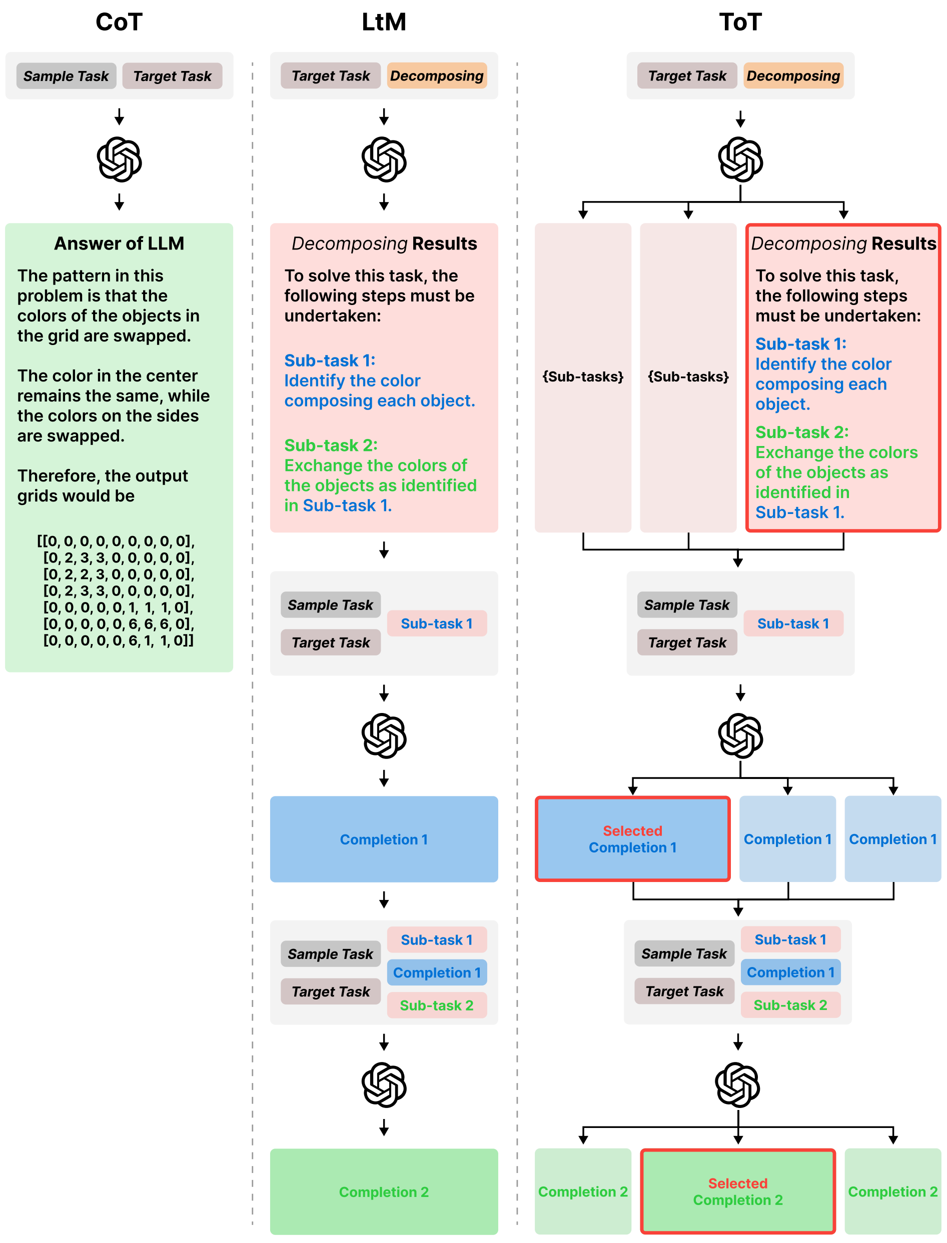

CoT
LtM
ToT
Sample Task
Target Task
Decomposing
Answer of LLM
The pattern in this problem is that the colors of the objects in the grid are swapped.
The color in the center remains the same, while the colors on the sides are swapped.
Therefore, the output grids would be
[[0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 2, 3, 3, 0, 0, 0, 0, 0], [0, 2, 2, 3, 0, 0, 0, 0, 0], [0, 2, 3, 3, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 1, 1, 1, 0], [0, 0, 0, 0, 0, 6, 6, 6, 0], [0, 0, 0, 0, 0, 6, 1, 1, 0]]
Decomposing Results
To solve this task, the following steps must be undertaken:
Sub-task 1: Identify the color composing each object.
Sub-task 2: Exchange the colors of the objects as identified in Sub-task 1.
{Sub-tasks}
{Sub-tasks}
Sub-task 1
Completion 1
Selected Completion 1
Sub-task 2
Completion 2
Selected Completion 2

Fig. 5. Grey blocks illustrate prompt sets delivered to the LLM, including the sample task, target task, and LLM's prior responses, as shown in Fig. 4. Green blocks denote the final answer. CoT relies on a single grey block, indicating that the LLM strictly follows the provided sub-tasks. Conversely, LtM and ToT prompt the LLM to generate and address sub-tasks sequentially, represented by decomposed results (red) and intermediate responses (blue). ToT further distinguishes itself from LtM by evaluating multiple suggestions for sub-task handling and selecting the most effective one through a voting mechanism.

3.1.3 Inferential Coherence of LLMs. In our second experiment, we tested the inferential coherence of LLMs, which measures their ability to maintain the same logical inference across tasks sharing an underlying analogical rule. To assess this, we examined whether the LLM could solve new problems defined by the same rules as previously solved ARC tasks.

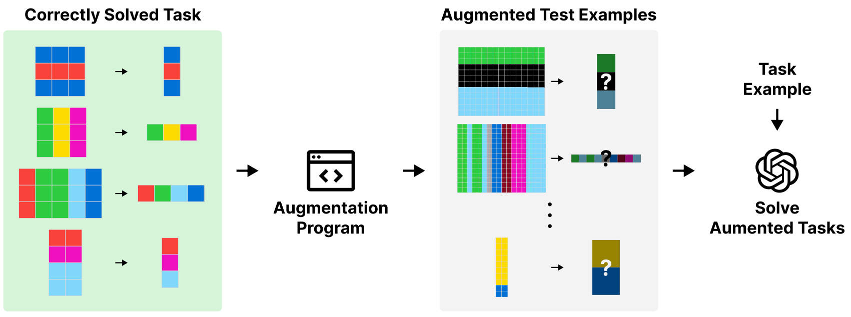


Fig. 6. Inferential coherence testing via augmentation: Using the Re-ARC augmentation program [23], we generated 100 new test examples per task. These examples retained the original analogical rule, allowing us to evaluate LLMs' inferential coherence across varied instances.

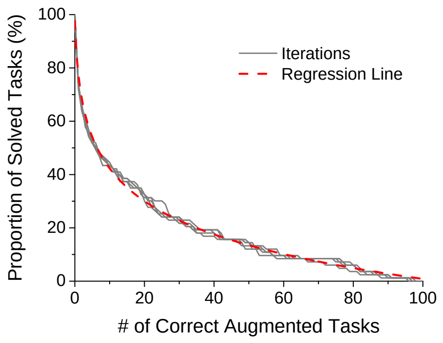
Algorithm 1 Calculating Inferential Coherence

Input: Training Tasks $\mathcal{T}$, LLM Model L(), Re-ARC Program $R()$, CoT Prompt P()
Output: Inferential Coherence of LLM C

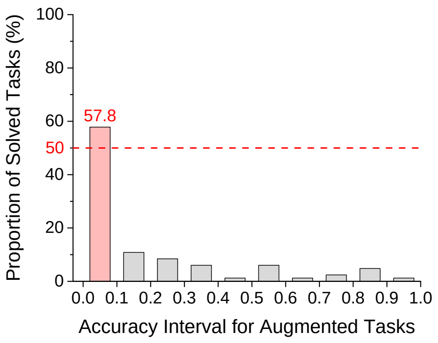
```
/* Step 1: Identify Solved Tasks $\mathcal{T}^S$ */
$\mathcal{T}^S \leftarrow \emptyset$                                  // $\mathcal{T}^S$ is a set of solved tasks by LLM
for $T_i \in \mathcal{T}$ do
    for $j \leftarrow 1$ to 5 do
        if $L(P(T_i)) == True$ then
            $\mathcal{T}^S \leftarrow \mathcal{T}^S \cup \{T_i\}$          // Consider $T_i$ is solved if LLM solves in 5 tries
            break

/* Step 2: Augment Solved Tasks $\mathcal{T}^A$ */
$\mathcal{T}^A \leftarrow \emptyset$                                  // $\mathcal{T}^A$ is a set of $\mathcal{T}_i^A$ generated below
for $T_i^S \in \mathcal{T}^S$ do
    $\mathcal{T}_i^A \leftarrow \emptyset$                             // $\mathcal{T}_i^A$ is a set of augmented tasks from $T_i^S$
    for $j \leftarrow 1$ to 100 do
        $\mathcal{T}_i^A \leftarrow \mathcal{T}_i^A \cup \{R(T_i^S)\}$                       // Augment $T_i^S$ using Re-ARC
    $\mathcal{T}^A \leftarrow \mathcal{T}^A \cup \mathcal{T}_i^A$

/* Step 3: Calculate Inferential Coherence $C$ */
$C \leftarrow [c_1, c_2, \ldots, c_{|\mathcal{T}^A|}]$          // $C$ is a list of the number of solved tasks in $\mathcal{T}_i^A$
for $\mathcal{T}_i^A \in \mathcal{T}^A$ do
    $c_i \leftarrow 0$
    for $T_{i,j}^A \in \mathcal{T}_i^A$ do
        if $L(P(T_{i,j}^A)) == True$ then
            $c_i \leftarrow c_i + 1$                                  // Count solved $T_{i,j}^A$ in $\mathcal{T}_i^A$

return $C$
```

Fig. 6 summarizes the experiment, and the detailed procedure is in Algorithm 1. We began by using GPT-4o to solve examples from 400 ARC tasks,[1] repeating this five times to identify consistently solvable tasks. For tasks solved correctly at least once, we used Re-ARC [23] to generate 100 additional examples that mirror the original approach. We hypothesized that a model demonstrating inferential coherence would solve all augmented examples, allowing us to rigorously test its generalization ability across similar tasks.

Fig. 7 presents two key analyses of the results. The cumulative distribution (Fig. 7a) shows consistent exponential decay patterns across all five iterations, indicating persistent low coherence regardless of iteration. The accuracy distribution (Fig. 7b) reveals that 57.8% of tasks achieved below 10% accuracy on augmented examples. Together, these results demonstrate LLMs' limited inferential coherence on ARC tasks.



(a) Cumulative distributions of successful solutions across 100 augmented examples per task. The y-axis shows the cumulative proportion of tasks achieving each success rate. The consistent exponential decay across five iterations demonstrates persistent low coherence.



(b) Distribution of ARC tasks by accuracy intervals of 0.1 on augmented examples. The y-axis shows the proportion of tasks per accuracy interval, averaged over five iterations. More than half (57.8%) of tasks achieved below 10% accuracy, demonstrating limited inferential coherence.

Fig. 7. Analysis of GPT-4o's performance on augmented test examples. For each of the 83 tasks successfully solved from the ARC training set, we generated 100 additional examples per task using Re-ARC [23] to evaluate inferential coherence.

3.1.4 Case Study: Semantic Coherence of LLMs. Finally, we analyzed how LLMs solved tasks in the two experiments described in Section 3.1.2 and Section 3.1.3. When evaluating not only the answers but also the process for the three prompts CoT, LtM, and ToT, we found that regardless of the prompt, the accuracy was about 3%, indicating that correct answers were being derived from incorrect processes, as shown in Fig. 8.

To solve the task, 1) identify 5×5 objects within the input grid, 2) count the number of black squares in each object, and 3) extract the object with the highest number of black squares. However, CoT, LtM, and ToT attempted to solve the task incorrectly. For CoT, objects in the input grid were sorted, and then the object in the middle was selected. Although CoT arrived at the correct answer, the method of sorting the objects lacked clarity. For LtM and ToT, there was an understanding that a specific object from the input grid needed to be selected to solve the task, but they mistakenly

[1]We selected 400 tasks from the ARC training set, as Re-ARC can only augment the training set

recognized objects from the test input grid. These solutions share a common flaw: they fail to establish a logically consistent rule across the different examples of training inputs and outputs provided. In other words, regardless of the prompting technique (CoT, LtM, or ToT), LLMs still struggle to demonstrate logical coherence in deriving a single rule that consistently applies across the examples given to solve the task.

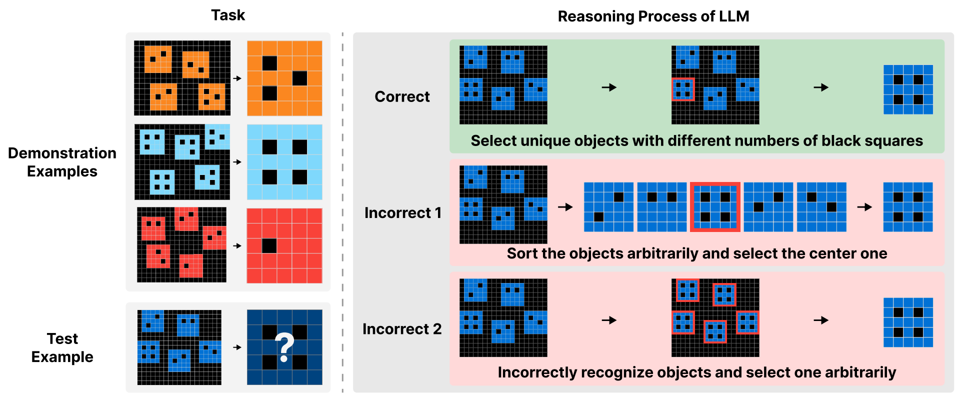

Task
Reasoning Process of LLM
Demonstration Examples
Test Example
?
Correct
Select unique objects with different numbers of black squares
Incorrect 1
Sort the objects arbitrarily and select the center one
Incorrect 2
Incorrectly recognize objects and select one arbitrarily

Fig. 8. Presenting instances where LLMs reach the correct answer but use flawed reasoning, highlighting the challenge of applying consistent logical rules across different ARC tasks. The task involves identifying a unique 5×5 object within a grid based on the number of black squares. The 'Correct' process shows the LLM correctly identifying the unique object, while 'Incorrect 1' and 'Incorrect 2' represent failed reasoning—one due to arbitrary selection and the other due to misidentification.

The inconsistency of inferring correct results from incorrect processes was also observed in the second experiment conducted on the training set. Upon analyzing the natural language explanations for the 83 tasks solved at least once out of 400 training tasks, we found that in 35 of these cases, the solutions proposed by the LLM could not produce the correct answer. This finding suggests that LLMs lack semantic coherence regardless of the prompting technique or tasks. In other words, LLMs derive outcomes unrelated to their reasoning process, as evidenced by generating correct answers from incorrect solutions.

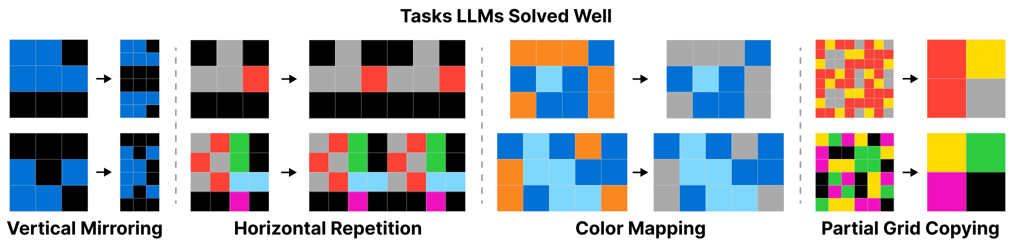


Fig. 9. Types of tasks where the LLM showed high accuracy. The LLM showed high accuracy in simple tasks such as pattern mirroring, pattern repetition, color mapping, and partial grid copying.

Nevertheless, in Section 3.1.3, we identified eight tasks that the LLM could solve with an accuracy of 0.6 or higher. As shown in Fig. 9, these eight tasks consist of simple solutions such as mirroring, color mapping, and partial grid copying. These tasks shared a common characteristic of conceptual simplicity, utilizing only one of the four prior knowledge domains included in ARC: objectness, goal-directedness, numbers and counting, and basic geometry [8]. For the 17 tasks that required the use of two or more prior knowledge domains, the LLM failed to solve any of the 100 augmented examples. The fact that the LLM could not solve any of the augmented examples, despite having solved the original ones, suggests that LLMs are not semantically coherent and may even indicate potential data leakage.

This comprehensive analysis demonstrates that while LLMs can solve certain simple pattern recognition tasks, they struggle with more complex reasoning that requires the integration of multiple concepts. The inability to coherently apply rules across augmented test examples, coupled with the generation of correct answers through incorrect reasoning, highlights significant limitations in both the inferential and semantic coherence of current LLM systems when tackling abstract reasoning tasks like those presented in ARC.

3.1.5 Conclusion. In Section 3.1, we evaluated the logical coherence of LLMs by solving 100 ARC tasks using three different prompting techniques. Our results, showing accuracies ranging from 4% to 12%, demonstrate variability in reasoning performance depending on the prompting approach. Additionally, when experimenting with GPT-4o on 400 training tasks, the LLM showed a high accuracy of 20%.

However, through an in-depth qualitative review, we demonstrated that the LLM's results may not be logically coherent. For the augmented test examples (100 for each solved task), the LLM only managed to achieve performance above 60% in eight out of the 83 solved tasks. Furthermore, for 35 out of the 83 solved tasks, nearly half of the solution processes provided by the LLM were incorrect and could not derive the correct results. This analysis suggests that the LLM has failed to achieve human-level logical coherence.

The results of this study align with previous research asserting that logical problem-solving remains challenging for LLMs alone. One study [60] found that LLMs can generate logically consistent reasoning with CoT prompting, even when their reasoning steps are flawed. Another study [77] showed that LLMs struggle with accurate self-reflection in tasks like mathematical reasoning and translation. Additionally, research [57] revealed that LLMs often fail to detect errors in intermediate steps, exposing flaws in their reasoning process. While these studies suggest that providing more context or enforcing stronger self-reflection might improve logical reasoning [60, 65, 77], our findings indicate that these challenges persist, suggesting the issue may not be simply a lack of information about the problem.

3.2 Capability of LLMs 2: Compositionality

3.2.1 Motivation. In Section 3.2, we investigate compositionality, the second concept of LoTH.[2] Compositionality refers to the ability to generate complex linguistic expressions given simpler ones [18]. This characteristic allows individuals to effectively tackle more complex tasks by breaking sub-tasks down into simpler steps, supporting the notion that humans can solve more complex tasks when faced with them. Strong compositionality enables the resolution of complex tasks and facilitates transparent descriptions of the process, which is also an important aspect of LLMs.

This section uses ARC to test the compositionality of LLMs. Previous studies have tested a model's compositionality by providing functions in the prompt that can be combined to solve tasks and then checking if the model can solve them [53]. Similarly, in this study, we also provide step-by-step functions, which we refer to as DSL (Domain Specific Language), and then conduct experiments to verify whether they can solve ARC tasks. Additionally, to understand why tasks might not be solved, we conducted further experiments on the model's comprehension of these

[2]While the Language of Thought Hypothesis principally uses the term 'systematicity', this study employs 'compositionality' as used in Fodor's paper. We use this term because compositionality encompasses a broader concept than systemicity.

functions. Therefore, we verify whether LLMs understand the meaning of the functions provided for ARC tasks and whether they can combine the functions appropriately to produce the desired results. The result of this experiment indicates that while LLMs sufficiently understand the functions and their relationship with images, their ability to decompose and combine functions to achieve the desired outcome is weak.

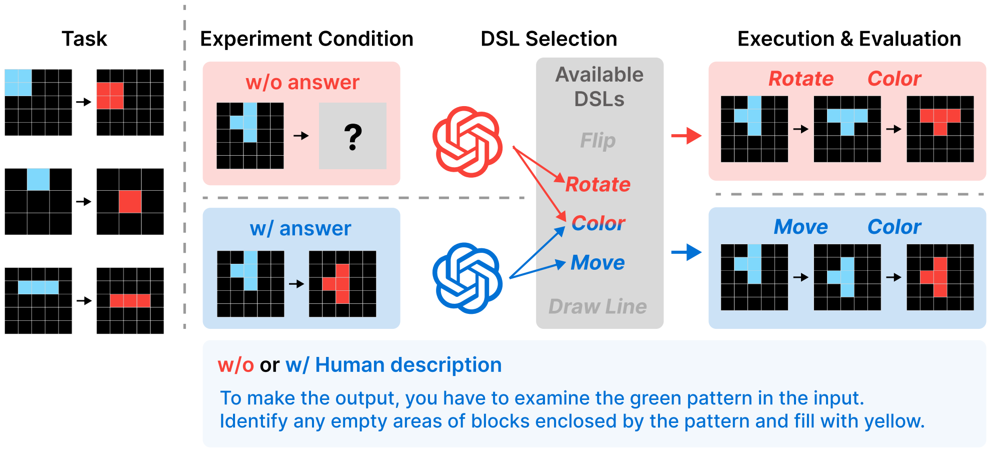

Task
Experiment Condition
DSL Selection
Execution & Evaluation
w/o answer
?
w/ answer
Available DSLs
Flip
Rotate
Color
Move
Draw Line
Rotate
Color
Move
Color
w/o or w/ Human description
To make the output, you have to examine the green pattern in the input.
Identify any empty areas of blocks enclosed by the pattern and fill with yellow.

Fig. 10. Overall process of DSL compositionality experiments. Before conducting the experiment, decisions are made on whether to provide 1) the test output and 2) a human description. During execution, the LLM analyzes the given demo examples to infer the rules and then selects the appropriate DSL steps from the DSL list to solve the test example. The chosen DSL steps are then applied to the test input grid within the DSL environment, which determines whether the answer is correct.

3.2.2 Compositionality of LLMs. In the first experiment, to measure compositionality, we provided LLM with information about DSL and asked them to solve given ARC tasks. Fig. 10 illustrates the structure of the entire experiment. If an LLM possesses sufficient compositionality, it should be able to select appropriate DSLs and their arguments for a given goal. However, in cases where the LLM failed to choose the correct DSL, we divided the conditions further to identify the cause. These conditions were whether the LLM understood the goal and the solution process. To analyze the results according to each condition, four types of experiments were conducted: 1) given only DSL, 2) given correct output along with DSL, 3) given human descriptions [51] to ARC test examples along with DSL, and 4) given both correct output grid and human descriptions along with DSL. Providing the correct output grid demonstrates compositionality based on knowing or not knowing the goal, while providing human descriptions shows the impact of natural language descriptions on compositionality.

We provided each DSL as a Python function. In this experiment, we used 19 types of DSL capable of solving ARC tasks. The prompts commonly included a brief explanation of ARC, DSL function code with comments, DSL usage examples, demonstration examples of tasks, inputs for the test examples, and object information of the test inputs. Object information is one of the crucial parameters in solving ARC tasks, which is why we added it to the prompt. We used the PnP algorithm [46] to extract object information from ARC tasks. The LLM returned a JSON-formatted string representing the chosen DSL and arguments at each step, which was used to verify whether the LLM reached the correct test output with an appropriate combination of DSL and arguments. We used the most recent model, GPT-4o, for this experiment.

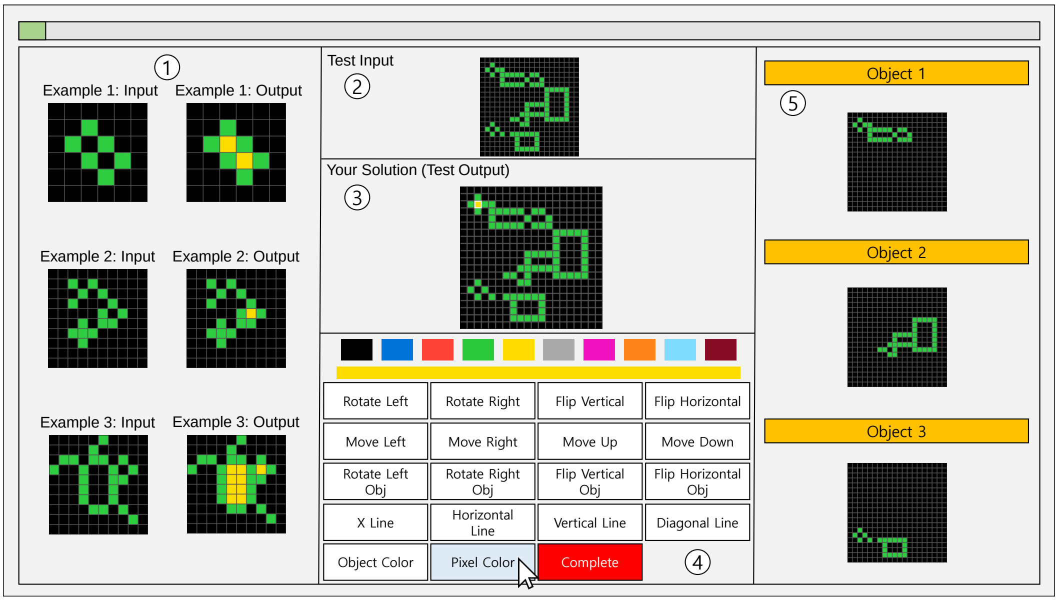


Fig. 11. Performance measurement tool for human participants: The interface consists of five numbered sections: 1) Demonstration examples from the given ARC task; 2) Test input grid for the given task; 3) Current grid for participants' responses; 4) Available DSLs; and 5) Object information automatically extracted using the PnP method. Participants must utilize the provided DSLs and object information to construct appropriate solutions, with both grid pixels and objects available as function parameters. Participants are meant to click complete if the current grid seems to be the solution.

Lastly, to establish a baseline, we conducted human experiments. We developed a specialized tool (Fig. 11) that provides the same information available to LLMs: ARC task demonstration examples, initial test input, current grid state, DSL functions, and object information extracted through PnP. Seven participants were constrained to solve the tasks using only the same DSLs given to the LLMs. Through these experiments, we identified that among the 800 publicly available ARC tasks, 158 tasks were solvable within 10 DSL steps using the given operations. Subsequently, all experiments in Section 3.2 were conducted on this subset of solvable tasks.

The experimental results are shown in Table 2. For LLM experiments, an average accuracy of 9% was observed when the test output was provided, and 3% without the test output. Compositionality strengthened when human explanations were included in the prompt, showing a similar improvement rate to the test output condition. Cronbach's alpha measurements showed consistency in responses, with all four experiments scoring above 0.7.

For human experiments, participants solved an average of 137 tasks, achieving approximately 86% accuracy on solvable tasks. This significant performance gap between LLMs (3–14%) and humans (86%) suggests that despite having access to the same information and tools, LLMs face fundamental challenges in DSL composition that humans can naturally overcome.

Table 2. Average accuracy from 10 repeated experiments based on the presence or absence of test output and human descriptions. The values in parentheses are Cronbach's alpha, and 'X' indicates that this condition was not applicable for human experiments as they performed the tasks without test outputs. In all the results in the table, Cronbach's alpha is greater than 0.7, indicating consistency.

	w/o Human Description	**w/ Human Description**	**Accuracy of Human**
w/o Test Output	3% (0.93)	8% (0.97)	86%
w/ Test Output	9% (0.96)	14% (0.96)	X

3.2.3 Analysis of compositional failures resulting from DSL misinterpretation. The issue is that the average accuracy described in Table 2 doesn't solely reflect compositionality. DSL provides a step-by-step manner to represent solution steps in ARC tasks. When we use a DSL to solve these tasks, we can think about the likelihood of choosing the right DSL for each step in two parts: 1) How well LLMs understand the DSL: This is reflected in how accurately it can predict the next grid when given the DSL instructions. 2) How necessary each predicted grid is in creating the final solution: This relates to how well the steps fit together to solve the task.

The overall chance of picking the correct DSL for all steps depends on both of these factors working together. To solve a task, all DSLs must be correct for 10 steps. Based on our preliminary analysis, we modeled this relationship as a multiplicative interaction between DSL understanding and compositional difficulty, as shown in Eq. 1. In this equation, n represents the DSL sequence length, w_n represents the number of tasks that need n steps to solve, p represents the single-step accuracy, and x represents the difficulty of composition for each task. We assumed that the LLM's compositionality could vary depending on the information provided to the LLM and the task.

$$y = \frac{\sum_{n=1}^{10} w_n \cdot (p \cdot x)^n}{\sum_{n=1}^{10} w_n} \tag{1}$$

To determine the task accuracy considering only the compositional difficulty, we must estimate the y value when $p = 1$. Therefore, we conducted an additional experiment, as shown in Fig. 12 to verify the probability of not finding an appropriate DSL due to the inability to predict the output grid when selecting a DSL.

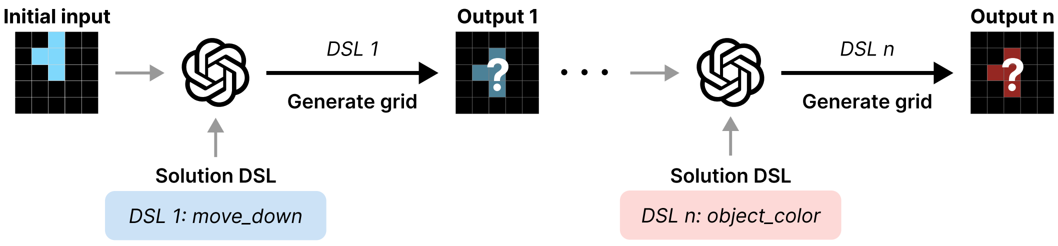


Fig. 12. Overall process of an experiment in understanding DSL. The task for the LLM is to accurately generate a grid transformed by the DSL when given a grid and its corresponding DSL. Each task involves a DSL sequence ranging from 1 to 10 steps, using trajectories previously solved by humans.

In the additional experiment, we focused on 158 tasks selected from the 800 publicly available ARC tasks, specifically choosing those that could be solved within 10 DSL steps. We checked how accurately the LLM could generate the correct output grid when given both the DSL and ARC input grid. Each task was repeated 10 times to ensure reliability. For these experiments, we provided the

LLM with correct DSL operations and argument chains created by human solvers. Among multiple human solutions, we prioritized those with the shortest step length to minimize complexity. Since both the input grid and DSL instructions were provided, LLM should be able to produce the correct output grid regardless of the sequence length, assuming perfect DSL comprehension.

The relationship between DSL sequence length and LLM's prediction accuracy is shown in Fig. 13. As the required sequence length increases, we observe a clear decline in the model's ability to predict correct output grids. Based on these observations, we calculated a weighted average single-step accuracy p using Eq. 2, where w_n represents the number of tasks with sequence length n, and a_n represents the prediction accuracy for that length. This calculation yielded an estimated single-step accuracy of 81%, indicating that errors compound significantly with longer sequences.

$$p = \frac{\sum_{n=1}^{10} w_n \cdot a_n}{\sum_{n=1}^{10} w_n} \tag{2}$$

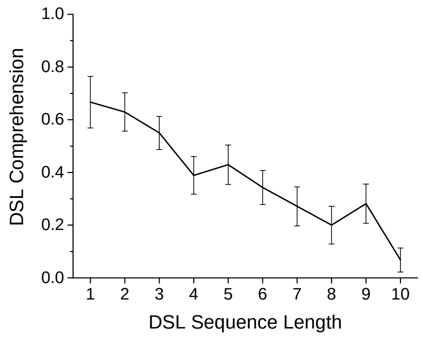


Fig. 13. Model's accuracy in predicting output grids for DSL sequences of varying lengths (1-10 operations). The y-axis shows the success rate of grid prediction after applying the given DSL operations. The DSL comprehension tends to decrease with longer sequences.

Table 3 presents the estimated accuracy when assuming perfect DSL understanding (p = 1.0, adjusted from the observed p = 0.8). This adjustment isolates the impact of compositional ability alone, showing that under ideal conditions with both test output and human descriptions provided, nearly 30% of tasks could be solved. The consistent 10 percentage point improvement observed when adding either the correct answer or natural language descriptions suggests that each element reduces the compositional difficulty (represented as x in Eq. 1) of the tasks.

Table 3. The table of results shows the accuracy estimates obtained using Eq. 1, assuming that the LLMs have a 100% understanding of DSL, meaning the single-step accuracy p is 1.0.

	w/o Human Description	**w/ Human Description**
w/o Test Output	5%	15%
w/ Test Output	17%	29%

3.2.4 *Case Study: Enhancement of Compositionality through Human Descriptions.*

One notable observation was the enhanced compositionality when human descriptions of problem-solving methods were included in prompts. To investigate how LLMs could solve tasks with human descriptions, we analyzed the solution processes of 13 additional tasks solved when human descriptions were provided. Results indicate that human descriptions facilitate task input and action abstraction, thereby improving problem-solving capabilities. For instance, LLMs fail to recognize patterns in the correct output without descriptions; however, they immediately identify patterns such as an 'X' shape with descriptions. These findings suggest the potential to enhance LLMs' reasoning performance by incorporating abstracted task information.

3.2.5 *Conclusion.*

In Section 3.2, experiments using ARC and DSL were conducted to measure the compositionality of LLMs. The results led to three conclusions. First, LLMs could predict the output grid when DSL was applied to the input with an average accuracy of about 81%. However, as the sequence length increased, the accuracy decreased, which appears to be due to cumulative errors. Second, when not given the correct answer, LLMs selected the correct DSL only 3% of the time, indicating a lack of ability both in inferring rules to predict the correct output grid and in selecting the appropriate DSL to reach the expected output. Finally, when human descriptions were added, the accuracy in choosing DSL increased to a level similar to when the correct answer was provided. Analysis of this process suggested that this improvement was due to linguistic abstraction of the ARC task and DSL combinations.

Previous studies have emphasized LLMs' limitations in combining simple elements to create new meanings, revealing struggles with compositionality. One study shows Transformers exhibit significant performance drops when tested on new function combinations, indicating challenges in systematically generalizing knowledge [25]. Another study introduced datasets like SADE to evaluate LLMs' ability to process visual and textual information, suggesting they still struggle with tasks like understanding negations and grasping complex content [38]. A further study examined how well LLMs can break down complex instructions or build them from simple ones. These found that while LLMs improve at understanding simple tasks by learning complex ones, they struggle with complex tasks when starting from simpler ones [73]. These findings across studies point to ongoing challenges in LLMs' ability to connect simple and complex elements, highlighting their compositionality limitations.

3.3 Capability of LLMs 3: Productivity

3.3.1 Motivation. In Section 3.3, we investigate the third concept of LoTH: productivity. Productivity refers to the ability to generate unseen representations based on observed data [18]. This characteristic enables humans to imagine diverse situations from a single phenomenon, facilitating efficient learning without the need for repetitive data exposure. Similarly, when endowed with this ability, LLMs are expected to excel in unseen tasks, making productivity a crucial function of essential reasoning. The capacity to generate new pairs within a constrained set of rules is particularly valuable for solving ARC tasks, highlighting the need for productivity. In this section, we will assess productivity by evaluating the validity of LLM-generated examples based on given example pairs from ARC tasks.

While productivity ideally involves testing for infinite generative capacity, practical limitations necessitate alternative approaches. The challenge lies in demonstrating that a system can produce an unlimited number of novel, meaningful outputs from a finite set of inputs and rules. Previous studies have addressed this challenge by examining whether valid outputs can be produced under added constraints [25, 31, 59]. These constraints serve to create a more manageable testing environment while still allowing for the assessment of generative capabilities. Following this methodology, our study investigates how effectively LLMs can generate valid outputs when presented with an ARC task and its underlying conceptual rule. This approach allows us to evaluate productivity within a controlled framework while still capturing the essence of generative capacity.

To understand how well LLMs can generate new expressions based on inherent logical concepts, we conduct experiments using ARC tasks. Productivity in this context involves two main steps: 1) inferring specific rules for image generation from example images and natural language expressions, and 2) applying these rules to generate new, unseen images. However, as explored in previous sections, the standard approach to solving ARC tasks is insufficient to confirm these two processes. Therefore, we propose a novel experiment: *Given an ARC task and a basic rule shared with similar ARC tasks, can LLMs generate valid examples of the given task?* If LLMs can understand the relationship between the given ARC task and the abstract rule, they should be able to infer specific rules for the task and generate new valid examples. Through this, we aim to determine whether LLMs can imitate the productivity of human thinking in generating novel solutions.

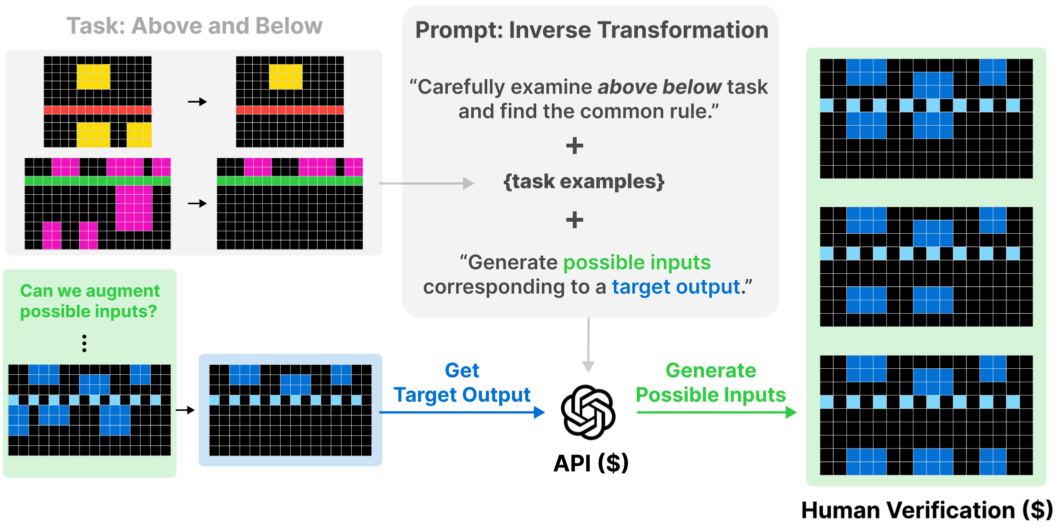
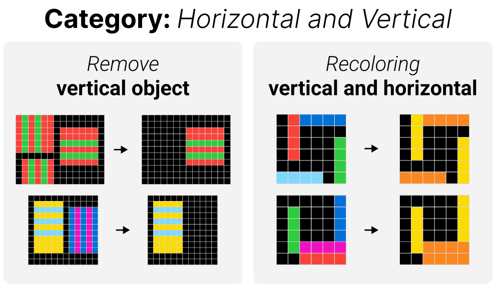

Task: Above and Below
Prompt: Inverse Transformation
"Carefully examine *above below* task and find the common rule."
+
{task examples}
+
"Generate possible inputs corresponding to a target output."
Can we augment possible inputs?
Get Target Output
API ($)
Generate Possible Inputs
Human Verification ($)

Fig. 14. Overall process of possible input generation with the Inverse Transformation Prompt (ITP). With ITP and one example of the task, LLMs generate input candidates of the output for the given example. If these generated inputs are valid, pairs created by these inputs and the given output can become new examples.

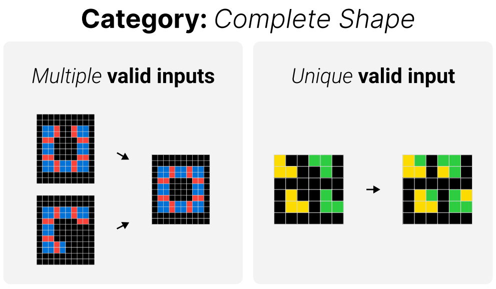
3.3.2 Validity of Augmentation. To evaluate whether LLMs can infer their own generation rules given ARC examples and create new tasks by appropriately applying these rules, we rigorously controlled the prompts. While ARC provides a diverse set of tasks, it lacks systematic categorization and explicit rules for each task. Therefore, we utilized ConceptARC [43], which maintains the same format as ARC but provides categories for each task, making it more suitable for our experimental design. We provided LLMs with two types of prompts: example pairs from the ConceptARC task and abstract rule descriptions applicable to similar tasks. In this step, one example pair served as the basis for generation, while the others were used to infer task-specific rules. Based on the ConceptARC framework, the tasks are organized into 16 distinct categories. For each category within ConceptARC, a corresponding abstract rule ensures that tasks within the same category adhere to the identical abstract rule.

We proposed the Inverse Transformation Prompting (ITP), a prompting technique for this experiment. ITP instructs LLMs to generate multiple valid examples by leveraging both the ConceptARC task and its associated abstract rules. Fig. 14 demonstrates how LLMs generate new examples, given the ConceptARC task and the corresponding ITP. Using this method, LLMs produce multiple inputs that can pair with the output from one example of the task. This example for generation is excluded from the ITP. If LLMs understood the ConceptARC task rules provided through ITP, the new example pairs generated by LLMs would be suitable as examples of the task.

ITP is based on a many-to-one approach to achieve two advantages. First, the input-only generation method is more data-efficient than generating both input and output, as existing task outputs can be reused without modification. Since all tasks in ConceptARC have example pairs, reusing these examples fully utilizes the given data. ITP allows a single ConceptARC task to be reused multiple times. In particular, using ITP can further increase data efficiency by allowing one ConceptARC task to be reused multiple times by changing the order of examples. Secondly, ITP increases the likelihood of generating valid responses. Through simulations, we observed that inferring inputs from outputs is more likely to generate valid results than the reverse. Because generating input from output is subject to relatively fewer constraints, there is a wide range of acceptable outcomes.


Category: Horizontal and Vertical
Remove vertical object
Recoloring vertical and horizontal

(a) Even within the same category, tasks can showcase varied objectives and complexities. The left task requires removing vertically striped objects, while the right focuses on recoloring objects based on their orientation.



(b) Depending on the task, there may be multiple or a unique input for an output. The left shows a task of completing a square with various inputs, and the right combines specific shapes, leading to a unique input.

Fig. 15. There are two challenges when LLMs generate examples through ITP: (a) task diversity within categories and (b) inflexibility in task-specific examples. These may cause difficulties in the process of LLMs generating examples through ITP.

In the process of creating ITP, we encounter two challenges. First, according to the ConceptARC category, tasks within the same category can have different specific objectives. Fig. 15a illustrates that there are various types of tasks with the same category. For example, even within the same category, the core solution for one task might involve removal, while another might focus on recoloring. This variation highlights that abstract rules given in the same sentences for each category may not be sufficient to cover various types of tasks. Second, there were ConceptARC tasks that made it impossible to infer multiple inputs from a single output (Fig. 15b). In such cases, there was only one valid input. Although we tried to take these cases into account while writing the ITP, these challenges nevertheless harmed the experimental results.

Before analyzing the experimental results, it was necessary to redefine the evaluation metric to reflect a shift in focus from solving tasks to generating valid examples. As previously explained, for a given example of a particular task, we generated valid inputs that could be paired with the corresponding output. To successfully generate these inputs, the LLM must derive the task's specific rules through its ITP and apply them to the output to create valid inputs. In this experiment, we evaluated whether all generated inputs were valid for each task. This metric assesses both the LLM's understanding of the correct rules and its ability to generate valid examples based on those rules. Consequently, this experiment systematically evaluates the LLMs' capability to generate logical and valid demo pairs, enhancing our understanding of their ability to create new representations.

Based on 160 ConceptARC tasks, we evaluated the validity of 2,913 generated examples. The average valid generation ratio was approximately 17.1%, with the remaining examples deemed invalid. As previously mentioned, the validity of the generated examples was determined by human judgment, assessing whether the generated tasks adhered to the analogical rules required to solve the task. The results in Table 4 show that LLMs exhibit a degree of capability in generating examples that align with the specified rules. However, there is a limitation due to weak criteria for

Table 4. The ratio of valid examples among examples generated for each category of ConceptARC.

Category	Generated	Valid	Validity
Above Below	158	34	21.52%
Center	236	35	14.83%
Clean Up	183	83	45.36%
Complete Shape	147	37	25.17%
Copy	153	4	2.61%
Count	202	29	14.36%
Extend To Boundary	167	8	4.79%
Extract Objects	176	21	11.93%
Filled Not Filled	203	29	14.29%
Horizontal Vertical	114	7	6.14%
Inside Outside	191	24	12.57%
Move To Boundary	165	12	7.27%
Order	162	26	16.05%
Same Different	246	76	30.89%
Top Bottom 2D	255	59	23.14%
Top Bottom 3D	215	25	11.63%
Total	2,913	509	17.12%

determining validity: even if infinite results can be generated, they cannot be reliably used without post-processing the data.

3.3.3 Case Study: Invalid Production. We analyzed the generated inputs to investigate the reasons behind LLMs' inability to produce valid inputs for ConceptARC tasks. Two major limitations were observed when LLMs generated new ConceptARC tasks. First, LLMs tended to simply copy inputs rather than infer meaningful rules from given example pairs. As shown in Fig. 16, this occurred repeatedly despite attempts to prevent it through prompts. Second, LLMs failed to properly consider the steps needed to generate inputs from given outputs. This frequently resulted in the creation of examples that could not be solved by the specific rules of the task. For instance, in cases where all vertices of a square were erased in the input, it became impossible to determine the color of the vertices, making it infeasible to infer the given output. These limitations suggest that LLMs lack an understanding of the semantics applicable to ConceptARC tasks and the ability to compose these semantics according to constraints.

3.3.4 Conclusion. In Section 3.3, we conducted experiments to confirm the productivity of LLMs by assessing whether they can understand given tasks in abstracted representations and generate valid new examples based on abstracted rules. Although it is known that LLMs have great strengths in creating creative works, our experimental results reveal that LLMs are weak in understanding rules and producing creations that adhere to those rules. Moreover, the observed limitations highlight a critical gap in LLMs' ability to engage in higher-level reasoning and abstraction, which are essential for successfully solving tasks that require an understanding of underlying principles rather than surface patterns. These results suggest that when LLMs generate outputs, they tend to mimic human-created results rather than truly understanding and applying rules. This makes it difficult for LLMs to reach the level of generation that humans can achieve.

Similarly, previous studies have shown similar results in measuring the productivity of AI models. Researchers tested how well pre-LLM models generalize to novel command combinations [31, 59]. Their findings revealed strong performance on trained data but weaknesses in generating

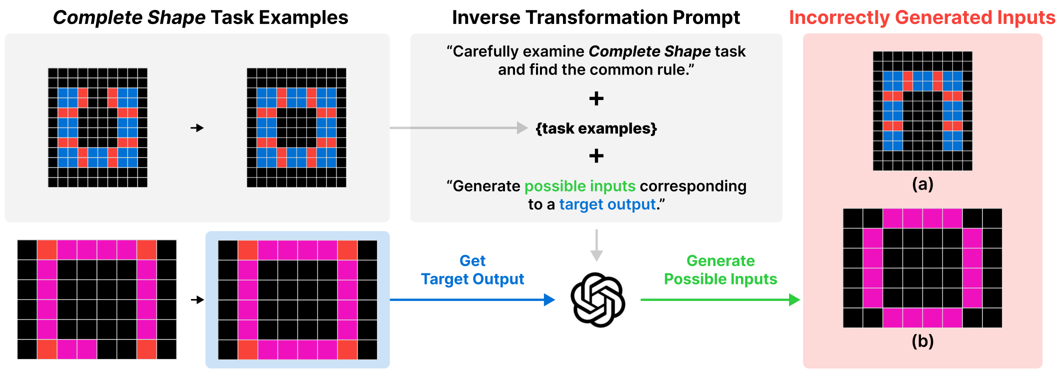

Complete Shape Task Examples
Inverse Transformation Prompt
Incorrectly Generated Inputs
"Carefully examine *Complete Shape* task and find the common rule."
+
{task examples}
+
"Generate possible inputs corresponding to a target output."
Get Target Output
Generate Possible Inputs
(a)
(b)

Fig. 16. Two incorrect generations for the task of completing the square shape: (a) The LLM generates the input from another example's output, and (b) the input does not provide enough information to infer the square's corner colors.

responses to unseen commands. Some researchers argued that LLMs struggle with generation under complex constraints and proposed improved models to address this issue [29, 35]. They propose novel frameworks to enhance LLMs for generating desired outputs when complex constraints are introduced, rather than relying solely on the base models. These researches share similarities with our study, which encountered difficulties in augmenting valid tasks based on complex rules.

4 Discussion

Through the three experiments in Section 3, we have observed that LLMs demonstrate strengths in understanding and manipulating both image and text inputs. However, they still exhibit weaknesses in logical inference, sequential planning based on understanding, and generating unseen images according to predefined rules. We will conclude by introducing the current research directions aimed at further enhancing LLMs' ability and outlining the goals after solving ARC.

4.1 What Should LLMs Possess to Solve ARC?

Based on the experimental results of Section 3, it is evident that LLMs still cannot solve ARC effectively. This is attributed to the deficiencies in logical coherence, compositionality, and productivity. How can we improve the inference capabilities of LLMs? In this section, we explore directions to enhance LLMs from the perspectives of abstraction knowledge and reasoning.

4.1.1 Abstract Knowledge. To solve ARC, the first challenge lies in extracting its implicit information. Xu et al. [70] emphasized the importance of object-based representation and proposed ARGA, which transforms example grids into graphs. Their follow-up study [71] demonstrated notable performance in object-based ARC tasks by leveraging ARGA-generated information. However, these methods have a critical limitation: they are inapplicable to ARC tasks without objects. Since only about 40% of ARC tasks involve object concepts [70], this approach cannot address more than half of the tasks. Wang et al. [64] partially enhanced LLM abstraction with a graph-form dataset, AbsPyramid, containing 221K textual descriptions, and proposed a framework called AbsInstruct. While structuring sentences can effectively abstract natural language, this approach is ineffective for ARC, which does not involve textual data.

4.1.2 Reasoning. Another challenge for LLMs in ARC is the vast search space. A promising approach involves enabling LLMs to generate DSLs themselves. Rajani et al. [48] introduced

CAGE, which prompts LLMs to generate explanations before generating answers. Subsequently, Wang et al. [62] reported improved results by having LLMs generate DSLs based on hypotheses they set themselves. Additionally, active research is underway on prompting techniques applying algorithmic approaches. Zhou et al. [79] demonstrated enhanced inference performance in LLMs by applying in-context learning. Follow-up research is being conducted following CoT and ToT. For example, CoT-SC [63] is a study that selects results through voting from multiple instances of CoT, GoT [3] secures flexibility by enabling the generation of graph-like thought nodes, and XoT [14] uses the thought tree while Monte Carlo tree search and refines the tree with reinforcement learning. However, these attempts are closer to additional learning for LLMs, and more research is needed to ascertain whether fundamental improvements in LLMs' reasoning abilities are achievable.

4.2 Future Direction After Solving ARC

Solving ARC tasks does not directly imply achieving human-level artificial intelligence. Moreover, there is a challenge in comparing task-solving approaches with those of humans. Thus, we suggest three alternatives to more accurately measure human-level inference abilities.

4.2.1 Using Different Benchmarks. One limitation of ARC is its simple environment. SQA3D [39], for instance, addresses inference tasks in a 3D domain by extending them into question-answering tasks using simulators like ScanNet [12]. Additionally, benchmarks such as TGIF-QA [26], MovieQA [55], TVQA [32], and STAR [69], which append question-answering to videos, have been proposed. Such benchmarks mimicking real-world inference scenarios could serve as supplements to measure complex abstractions not covered by ARC.

4.2.2 Quantification of ARC Task-Solving Processes. Chollet, the creator of ARC, argued that ARC maximizes generality while minimizing prior and experience [8], but these components have not been quantitatively evaluated. As a result, the quantitative assessment of factors such as the generality achieved by models solving ARC, the level of prior knowledge, and the components of prior knowledge remains elusive. One possible way to quantitatively evaluate the process of solving ARC tasks is to quantify the model's achievement of prior, experience, and generality.

4.2.3 Adding Evaluation Methods to Compare Task-Solving Processes with Human Approaches. Recent ARC research has focused on finding ways for AI to solve tasks. However, there are doubts about how similar these solutions are to those of humans. The initial paper by Johnson et al. [27] analyzed human ARC solutions. Subsequently, LARC [1] was proposed to analyze how tasks are solved through the language-based explanation of human solutions. Tools for facilitating the collection of human data are also continuously being developed. Kim et al. [28], for instance, have analyzed how tasks are solved through O2ARC. Based on these studies, we suggest calculating each ARC task's correctness and adding similarity with human data to the evaluation.

4.3 Recent Research Trends on the Reasoning Abilities of LLMs

In this paper, we utilized the ARC to evaluate and enhance the reasoning capabilities of LLMs. ARC serves as a crucial benchmark for testing AI models' ability to perform human-like reasoning. Beyond ARC, datasets such as DROP [15], CommonsenseQA [54], BoolQ [10], and GSM8K [11] provide invaluable resources to enhance the diverse reasoning capabilities of LLMs.

Recent studies indicate that LLMs still exhibit significant limitations in their reasoning abilities despite their proficiency in language-based tasks. LLMs still exhibit significant limitations in their reasoning abilities. Carvalho et al. [13] found that LLMs struggle with reasoning and decision-making in tasks beyond their training data, particularly in non-linguistic tasks requiring strategic thinking and spatial reasoning. Similarly, Gendron et al. [21] revealed poor performance on tasks requiring the identification and application of general patterns from limited examples. These studies collectively highlight that current LLMs, though advanced in linguistic tasks, are still far from achieving robust reasoning abilities across diverse domains.

To address these limitations, several advanced approaches have been developed. These include reinforcement learning with human feedback [9], CoT prompting [65], reasoning-centric fine-tuning [33], incorporating knowledge graphs during pre-training [36], and explainable AI techniques [4]. These approaches play a crucial role in advancing LLMs' reasoning capabilities across various domains.

Moreover, recent research has introduced innovative approaches to further augment the reasoning capabilities of LLMs. These include multimodal learning techniques [52], adaptive learning strategies with human feedback [45], and integration of programming languages with LLMs [19]. These cutting-edge studies significantly contribute to systematically strengthening the multidimensional reasoning capabilities of LLMs.

5 Conclusions

This study addresses the limitations of result-oriented analysis in LLMs' reasoning abilities by adopting the Language of Thought Hypothesis (LoTH). While recent LLMs have shown performance levels close to humans, experiments reveal significant gaps in planning and reasoning. Through LoTH's three components–logical coherence, compositionality, and productivity–we provide a structured approach to evaluate the reasoning process rather than just outcomes.

Using the Abstraction and Reasoning Corpus (ARC) as our benchmark, we conducted three quantitative experiments:

(1) **Logical Coherence:** Our analysis revealed significant gaps in both inferential and semantic coherence. While LLMs occasionally produced correct answers, they frequently failed to maintain logical consistency across similar problems and often derived correct results through flawed reasoning processes.
(2) **Compositionality:** LLMs showed fundamental limitations in combining simple components to solve complex problems. Their performance degraded significantly with increasing task complexity, and they struggled with DSL selection even when provided with additional context, indicating weak compositional abilities.
(3) **Productivity:** LLMs demonstrated significant weaknesses in a rule-based generation despite their known capabilities in creative tasks. They often resorted to mimicking observed patterns rather than truly understanding and applying abstract rules to generate valid new examples.

These findings suggest that current LLMs, despite their impressive performance metrics, lack fundamental reasoning capabilities when evaluated from a process-oriented perspective. To advance toward human-level artificial intelligence, future research should pursue three complementary directions. First, LLMs need enhancement in both abstraction knowledge and reasoning capabilities: this could involve developing better representation methods for implicit information extraction and exploring advanced prompting techniques to handle vast search spaces efficiently. Second, to ensure meaningful progress, we need to develop more comprehensive evaluation frameworks that can: (1) incorporate diverse benchmarks that better reflect real-world reasoning scenarios; (2) quantitatively measure solution processes beyond mere task completion; and (3) enable systematic comparisons between AI and human reasoning approaches. This study ultimately contributes to the field by providing a structured framework for evaluating and advancing AI reasoning capabilities, highlighting the importance of aligning AI development with human cognitive processes.

Acknowledgments

This work was supported by the IITP (RS-2023-00216011, RS-2024-00445087, No. 2019-0-01842) and the NRF (RS-2024-00451162) grants funded by the Ministry of Science and ICT, Korea. Experiments are supported by the Accelerate Foundation Models Research Initiative, Microsoft.

References

[1] Samuel Acquaviva, Yewen Pu, Marta Kryven, Theodoros Sechopoulos, Catherine Wong, Gabrielle Ecanow, Maxwell Nye, Michael Tessler, and Joshua B. Tenenbaum. 2022. Communicating Natural Programs to Humans and Machines. In *NeurIPS*.

[2] Zeyuan Allen-Zhu and Yuanzhi Li. 2023. Physics of Language Models: Part 3.2, Knowledge Manipulation. *arXiv:2309:14402* (2023).

[3] Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Michal Podstawski, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. 2023. Graph of Thoughts: Solving Elaborate Problems with Large Language Models. *arXiv:2308.09687* (2023).

[4] Or Biran and Courtenay Cotton. 2017. Explanation and Justification in Machine Learning: A Survey. In *IJCAI Workshop*.

[5] Alexey Borsky. 2021. ARC-Game. https://github.com/volotat/ARC-Game

[6] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang. 2023. Sparks of Artificial General Intelligence: Early Experiments with GPT-4. *arXiv:2303.12712* (2023).

[7] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, Wei Ye, Yue Zhang, Yi Chang, Philip S. Yu, Qiang Yang, and Xing Xie. 2024. A Survey on Evaluation of Large Language Models. *ACM Transactions on Intelligent Systems and Technology* (2024).

[8] François Chollet. 2019. On the Measure of Intelligence. *arXiv:1911.01547* (2019).

[9] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep Reinforcement Learning from Human Preferences. In *NeurIPS*.

[10] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. BoolQ: Exploring the Surprising Difficulty of Natural Yes/No Questions. In *NAACL*.

[11] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training Verifiers to Solve Math Word Problems. *arXiv:2110.14168* (2021).

[12] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. 2017. ScanNet: Richly-Annotated 3D Reconstructions of Indoor Scenes. In *CVPR*. 5828–5839.

[13] Gonçalo Hora de Carvalho, Robert Pollice, and Oscar Knap. 2024. Show, Don't Tell: Evaluating Large Language Models Beyond Textual Understanding with ChildPlay. *arXiv:2407.11068* (2024).

[14] Ruomeng Ding, Chaoyun Zhang, Lu Wang, Yong Xu, Minghua Ma, Wei Zhang, Si Qin, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. 2023. Everything of Thoughts: Defying the Law of Penrose Triangle for Thought Generation. *arXiv:2311.04254* (2023).

[15] Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. DROP: A Reading Comprehension Benchmark Requiring Discrete Reasoning Over Paragraphs. In *NAACL*.

[16] Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Sean Welleck, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. 2023. Faith and Fate: Limits of Transformers on Compositionality. In *NeurIPS*.

[17] Jerry A Fodor. 1975. *The Language of Thought*. Vol. 5. Harvard University Press.

[18] Jerry A Fodor and Zenon W Pylyshyn. 1988. Connectionism and Cognitive Architecture: A Critical Analysis. *Cognition* 28, 1-2 (1988), 3–71.

[19] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. PAL: Program-Aided Language Models. In *ICML*. 10764–10799.

[20] Howard E Gardner. 2011. *Frames of Mind: The Theory of Multiple Intelligences*. Basic Books.

[21] Gaël Gendron, Qiming Bao, Michael Witbrock, and Gillian Dobbie. 2024. Large Language Models are Not Strong Abstract Reasoners. *IJCAI* (2024).

[22] Wes Gurnee and Max Tegmark. 2023. Language Models Represent Space and Time. *arXiv:2310.02207* (2023).

[23] Michael Hodel. 2024. Addressing the Abstraction and Reasoning Corpus via Procedural Example Generation. *arXiv:2404.07353* (2024). https://github.com/michaelhodel/re-arc?tab=readme-ov-file

[24] Jie Huang and Kevin Chen-Chuan Chang. 2023. Towards Reasoning in LLMs: A Survey. In *ACL Findings*.

[25] Dieuwke Hupkes, Verna Dankers, Mathijs Mul, and Elia Bruni. 2020. Compositionality Decomposed: How Do Neural Networks Generalise? *Journal of Artificial Intelligence Research* 67 (2020), 757–795.

[26] Yunseok Jang, Yale Song, Youngjae Yu, Youngjin Kim, and Gunhee Kim. 2017. TGIF-QA: Toward Spatio-Temporal Reasoning in Visual Question Answering. In *CVPR*. 2758–2766.

[27] Aysja Johnson, Wai Keen Vong, Brenden M. Lake, and Todd M. Gureckis. 2021. Fast and Flexible: Human Program Induction in Abstract Reasoning Tasks. In *CogSci*.

[28] Subin Kim, Prin Phunyaphibarn, Donghyun Ahn, and Sundong Kim. 2022. Playgrounds for Abstraction and Reasoning. In *NeurIPS Workshop on nCSI*.

[29] Jing Yu Koh, Daniel Fried, and Russ R Salakhutdinov. 2024. Generating Images with Multimodal Language Models. In *NeurIPS*, Vol. 36.

[30] Lab42. 2024. ARC-PRIZE Competition. https://arcprize.org/

[31] Brenden Lake and Marco Baroni. 2018. Generalization without Systematicity: On the Compositional Skills of Sequence-to-Sequence Recurrent Networks. In *ICML*. PMLR, 2873–2882.

[32] Jie Lei, Licheng Yu, Mohit Bansal, and Tamara L Berg. 2018. TVQA: Localized, Compositional Video Question Answering. In *EMNLP*.

[33] Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. 2022. Solving Quantitative Reasoning Problems with Language Models. In *NeurIPS*.

[34] Zhaoyi Li, Gangwei Jiang, Hong Xie, Linqi Song, Defu Lian, and Ying Wei. 2024. Understanding and Patching Compositional Reasoning in LLMs. *arXiv:2402.14328* (2024).

[35] Mushui Liu, Yuhang Ma, Xinfeng Zhang, Yang Zhen, Zeng Zhao, Zhipeng Hu, Bai Liu, and Changjie Fan. 2024. LLM4GEN: Leveraging Semantic Representation of LLMs for Text-to-Image Generation. *arXiv:2407.00737* (2024).

[36] Weijie Liu, Peng Zhou, Zhe Zhao, Zhiruo Wang, Qi Ju, Haotang Deng, and Ping Wang. 2020. K-BERT: Enabling Language Representation with Knowledge Graphs. In *AAAI*.

[37] Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. 2024. MathVista: Evaluating Math Reasoning in Visual Contexts with GPT-4V, Bard, and Other Large Multimodal Models. In *ICLR*.

[38] Teli Ma, Rong Li, and Junwei Liang. 2024. An Examination of the Compositionality of Large Generative Vision-Language Models. In *NAACL*.

[39] Xiaojian Ma, Silong Yong, Zilong Zheng, Qing Li, Yitao Liang, Song-Chun Zhu, and Siyuan Huang. 2023. SQA3D: Situated Question Answering in 3D Scenes. In *ICLR*.

[40] Warren S McCulloch and Walter Pitts. 1943. A Logical Calculus of the Ideas Immanent in Nervous Activity. *The Bulletin of Mathematical Biophysics* 5 (1943), 115–133.

[41] Małkiński Mikołaj and Mańdziuk Jacek. 2023. A Review of Emerging Research Directions in Abstract Visual Reasoning. *Information Fusion* 91 (2023), 713–736.

[42] Suvir Mirchandani, Fei Xia, Pete Florence, Brian Ichter, Danny Driess, Montserrat Gonzalez Arenas, Kanishka Rao, Dorsa Sadigh, and Andy Zeng. 2023. Large Language Models as General Pattern Machines. *arXiv:2307.04721* (2023).

[43] Arseny Moskvichev, Victor Vikram Odouard, and Melanie Mitchell. 2023. The ConceptARC Benchmark: Evaluating Understanding and Generalization in the ARC Domain. *TMLR* (2023).

[44] Weili Nie, Zhiding Yu, Lei Mao, Ankit B Patel, Yuke Zhu, and Anima Anandkumar. 2020. Bongard-Logo: A New Benchmark for Human-Level Concept Learning and Reasoning. In *NeurIPS*.

[45] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simense, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training Language Models to Follow Instructions with Human Feedback. In *NeurIPS*.

[46] Jaehyun Park, Jaegyun Im, Sanha Hwang, Mintaek Lim, Sabina Ualibekova, Sejin Kim, and Sundong Kim. 2023. Unraveling the ARC Puzzle: Mimicking Human Solutions with Object-Centric Decision Transformer. *ICML Workshop on ILHF* (2023).

[47] Linlu Qiu, Liwei Jiang, Ximing Lu, Melanie Sclar, Valentina Pyatkin, Chandra Bhagavatula, Bailin Wang, Yoon Kim, Yejin Choi, Nouha Dziri, and Xiang Ren. 2024. Phenomenal Yet Puzzling: Testing Inductive Reasoning Capabilities of Language Models with Hypothesis Refinement. In *ICLR*.

[48] Nazneen Fatema Rajani, Bryan McCann, Caiming Xiong, and Richard Socher. 2019. Explain Yourself! Leveraging Language Models for Commonsense Reasoning. In *ACL*.

[49] Frank Rosenblatt. 1958. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review* 65, 6 (1958), 386–408.

[50] Stuart J Russell and Peter Norvig. 1995. *Artificial Intelligence: A Modern Approach.* Pearson.

[51] Donghyeon Shin, Seungpil Lee, Klea Lena Kovacec, and Sundong Kim. 2024. From Generation to Selection: Findings of Converting Analogical Problem-Solving into Multiple-Choice Questions. In *EMNLP Findings*.

[52] Amanpreet Singh, Ronghang Hu, Vedanuj Goswami, Guillaume Couairon, Wojciech Galuba, Marcus Rohrbach, and Douwe Kiela. 2022. FLAVA: A Foundational Language and Vision Alignment Model. In *CVPR*.

[53] Sania Sinha, Tanawan Premsri, and Parisa Kordjamshidi. 2024. A Survey on Compositional Learning of AI Models: Theoretical and Experimental Practices. *arXiv:2406.08787* (2024).

[54] Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. CommonsenseQA: A Question Answering Challenge Targeting Commonsense Knowledge. In *NAACL*.

[55] Makarand Tapaswi, Yukun Zhu, Rainer Stiefelhagen, Antonio Torralba, Raquel Urtasun, and Sanja Fidler. 2016. MovieQA: Understanding Stories in Movies through Question-Answering. In *CVPR*. 4631–4640.

[56] Alan Turing. 1950. Computing Machinery and Intelligence. *Mind* 59, 236 (1950), 433–460.

[57] Gladys Tyen, Hassan Mansoor, Victor Cărbune, Yuanzhu Peter Chen, and Tony Mak. 2024. LLMs Cannot Find Reasoning Errors, but Can Correct Them Given the Error Location. In *ACL Findings*.

[58] Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2024. PlanBench: An Extensible Benchmark for Evaluating Large Language Models on Planning and Reasoning about Change. In *NeurIPS*.

[59] Frank van der Velde, Gwendid T van der Voort van der Kleij, and Marc de Kamps. 2004. Lack of Combinatorial Productivity in Language Processing with Simple Recurrent Networks. *Connection Science* 16, 1 (2004), 21–46.

[60] Boshi Wang, Sewon Min, Xiang Deng, Jiaming Shen, You Wu, Luke Zettlemoyer, and Huan Sun. 2023. Towards Understanding Chain-of-Thought Prompting: An Empirical Study of What Matters. In *ACL*.

[61] Haoyu Wang, Mo Yu, Xiaoxiao Guo, Rajarshi Das, Wenhan Xiong, and Tian Gao. 2019. Do Multi-Hop Readers Dream of Reasoning Chains?. In *EMNLP Workshop on MRQA*.

[62] Ruocheng Wang, Eric Zelikman, Gabriel Poesia, Yewen Pu, Nick Haber, and Noah D Goodman. 2024. Hypothesis Search: Inductive Reasoning with Language Models. In *ICLR*.

[63] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *ICLR*.

[64] Zhaowei Wang, Haochen Shi, Weiqi Wang, Tianqing Fang, Hongming Zhang, Sehyun Choi, Xin Liu, and Yangqiu Song. 2023. AbsPyramid: Benchmarking the Abstraction Ability of Language Models with a Unified Entailment Graph. *arXiv:2311.09174* (2023).

[65] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *NeurIPS*.

[66] Peter West, Ximing Lu, Nouha Dziri, Faeze Brahman, Linjie Li, Jena D Hwang, Liwei Jiang, Jillian Fisher, Abhilasha Ravichander, Khyathi Chandu, Benjamin Newman, Pang Wei Koh, Allyson Ettinger, and Yejin Choi. 2024. The Generative AI Paradox: “What It Can Create, It May Not Understand”. In *ICLR*.

[67] Norbert Wiener. 1950. Cybernetics. *Bulletin of the American Academy of Arts and Sciences* 3, 7 (1950), 2–4.

[68] Johan Sokrates Wind. 2020. ARC-Solution. https://github.com/top-quarks/ARC-solution

[69] Bo Wu, Shoubin Yu, Zhenfang Chen, Joshua B Tenenbaum, and Chuang Gan. 2021. STAR: A Benchmark for Situated Reasoning in Real-World Videos. In *NeurIPS*.

[70] Yudong Xu, Elias B. Khalil, and Scott Sanner. 2023. Graphs, Constraints, and Search for the Abstraction and Reasoning Corpus. In *AAAI*.

[71] Yudong Xu, Wenhao Li, Pashootan Vaezipoor, Scott Sanner, and Elias B Khalil. 2023. LLMs and the Abstraction and Reasoning Corpus: Successes, Failures, and the Importance of Object-based Representations. *TMLR* (2023).

[72] Cheng Xue, Vimukthini Pinto, Chathura Gamage, Ekaterina Nikonova, Peng Zhang, and Jochen Renz. 2023. Phy-Q as a Measure for Physical Reasoning Intelligence. *Nature Machine Intelligence* 5, 1 (2023), 83–93.

[73] Haoran Yang, Hongyuan Lu, Wai Lam, and Deng Cai. 2024. Exploring Compositional Generalization of Large Language Models. In *NAACL Workshop*.

[74] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. In *NeurIPS*.

[75] Chi Zhang, Feng Gao, Baoxiong Jia, Yixin Zhu, and Song-Chun Zhu. 2019. RAVEN: A Dataset for Relational and Analogical Visual Reasoning. In *CVPR*.

[76] Jinghan Zhang, Xiting Wang, Weijieying Ren, Lu Jiang, Dongjie Wang, and Kunpeng Liu. 2024. RATT: A Thought Structure for Coherent and Correct LLM Reasoning. *arXiv:2406.02746* (2024).

[77] Wenqi Zhang, Yongliang Shen, Linjuan Wu, Qiuying Peng, Jun Wang, Yueting Zhuang, and Weiming Lu. 2024. Self-Contrast: Better Reflection Through Inconsistent Solving Perspectives. In *ACL*.

[78] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. 2023. Least-to-Most Prompting Enables Complex Reasoning in Large Language Models. In *ICLR*.

[79] Hattie Zhou, Azade Nova, Hugo Larochelle, Aaron Courville, Behnam Neyshabur, and Hanie Sedghi. 2022. Teaching Algorithmic Reasoning via In-Context Learning. *arXiv:2211.09066* (2022).

A Supplementary Analysis

A.1 Comparing LLM and Human Problem Difficulty Perception

Following the analysis in Section 3.1.4, we analyzed problems that LLMs (Large Language Models) solve well and those they struggle with. Table 5 presents the accuracy of LLMs across problem difficulty levels classified by humans. The classification was based on the existing categorization, relying on perceived difficulty by humans [5]. As a result, we discovered a tendency where problems perceived as difficult by humans align closely with those challenging for LLMs. Difficult problems shared two commonalities: 1) they required lengthy inference processes to solve, and 2) they involved considering multiple simultaneous problems to extract information about changes. An example from Fig. 17 illustrates this point: a task classified as 'Entry' only requires a single step of coloring, while a task classified as 'Hard' requires three steps: recognizing each object, identifying the priority of each object, and merging each object considering their priority. 'Easy' and 'Medium' are tasks that require relatively more complex steps than 'Entry' and fewer steps than 'Hard'. Considering these observations, it can be inferred that artificial intelligence possesses simple forms of visual logic that deal with only one of the four priors included in ARC: objectness, goal-directedness, numbers and counting, and basic geometry. However, it cannot handle complex combinations of logic that integrate these priors.

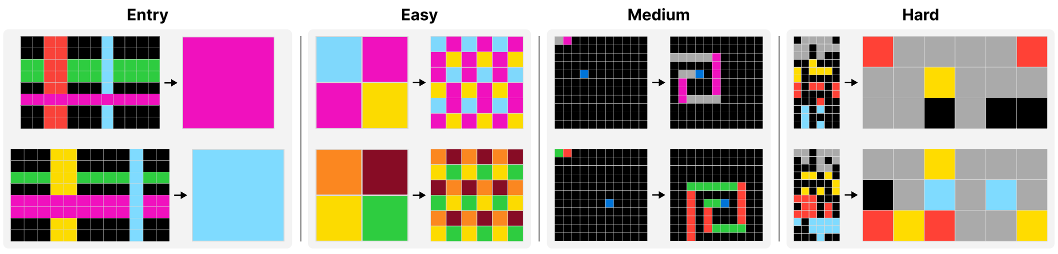


Fig. 17. Showcases of ARC tasks organized by human-perceived difficulty levels. These tasks illustrate the spectrum of complexity that humans use to rate problems, ranging from single-step 'Entry' level tasks to multi-step 'Hard' challenges. The difficulty classification reflects both the depth of inference required and the number of logical operations needed to reach a solution, paralleling the varying success rates of LLMs in tackling these tasks.

Table 5. Analyzing LLMs' reasoning capabilities by task difficulty, following prior categorization [5]. The number of ARC tasks corresponding to each category is listed in the table, and the experiment was performed five times for each task.

	Entry	Easy	Medium	Hard
Tasks	2	20	46	14
Trials	10	100	230	70
CoT	100.00%	30.00%	0.00%	0.00%
LtM	20.00%	19.00%	0.00%	2.85%
ToT	50.00%	22.00%	0.00%	0.00%
Average	56.67%	23.67%	0.00%	0.95%

A.2 Comparison of Augmentation Cost-Efficiency across GPT Versions

In a follow-up experiment to our productivity study, we aimed to compare the cost-efficiency of GPT-3.5 and GPT4-32k when augmenting demonstration example tasks. This investigation was crucial to understanding the trade-offs between model performance and associated costs in real-world applications.

Our experimental setup began with the creation of a prompt describing the category. Using this prompt, we developed an Inverse Transformation Prompt (ITP) and proceeded to augment demonstration examples using both GPT-3.5-16k and GPT-4-32k models. Throughout this process, we meticulously logged all prompts given to the LLMs and their corresponding responses.

To analyze the cost implications, we tokenized the logged text using the tiktoken library. We then calculated the cost of generating a single valid demonstration example based on the per-token cost specified by the Azure OpenAI API. This approach allowed us to accurately assess the financial implications of using each model for demonstration example augmentation. Validation of the generated examples was a critical component of our experiment. We employed human reviewers to manually verify the quality and appropriateness of the outputs. These reviewers were tasked with confirming two key aspects:

(1) Whether the results could be legitimately generated from the given rules.
(2) If the generated results were unique, avoid repetition or trivial variations.

This rigorous validation process ensured that our assessment of "valid" examples was thorough and meaningful in the context of practical applications.

Table 6. Comparison of augmentation cost-efficiency across GPT versions. The experiment was conducted on each demonstration example pair within the 16 task categories of Concept ARC. This table shows the results of LLMs generating valid demonstration example pairs and costs.

	Generated Example	**Valid Example**	**Validity**	**Cost per Valid Example**
GPT-3.5-16k	346	24	6.94%	\$0.0275
GPT-4-32k	411	40	9.73%	\$0.3925

Analysis of the cost to generate valid demonstration examples, as illustrated in Table 6, reveals that while GPT-4-32k showed approximately 1.5 times higher performance in terms of validity compared to GPT-3.5-16k, its cost was nearly 20 times higher. This suggests that productivity gains may not scale linearly with model capability and cost, especially when generating outputs under complex constraints. Consequently, in scenarios requiring valid outputs under intricate constraints, GPT-3.5 might be preferable to GPT4-32k when considering the trade-off between performance improvement and cost increase. However, the low overall validity rate of less than 10% for both models indicates that current LLMs still have significantly lower productivity compared to humans in such tasks. This finding suggests that merely upgrading to more advanced models is unlikely to fully resolve the productivity gap, highlighting the need for further research and development in enhancing LLM performance for complex, constrained tasks.

A.3 Limitations of ARC as a Benchmark for Human-Level AI

Does solving ARC signify the completion of human-like AI? To answer this question, two doubts need to be appropriately addressed: 1) Will the ARC solver possess human-level problem-solving abilities? and 2) Will that solver think like humans to solve ARC? It is not easy to imagine how the ARC solver operates without human-level reasoning. At this point, what we can assume is that the model will have the three properties of LoTH, and the model could be capable of several types of reasoning included in ARC. With this hypothesis, we attempt to address the following questions.

A.3.1 Will the Model Possess Human-Level Problem-Solving Abilities? Being capable of reasoning does not necessarily equate to having human-level problem-solving abilities. In other words, even if a model can reason to a level that can solve ARC, it may not have human-level problem-solving capabilities. Various tasks that humans face are generally more complex than ARC and involve various other cognitive factors besides reasoning. Therefore, even models that can solve ARC may have the following limitations compared to human-level problem-solving abilities.

First, with the current ARC criteria, it is still unknown whether the model that solved it can solve more complex types of tasks. This is because ARC tasks focus on just reasoning and are therefore presented in a relatively simple environment. Whether the reasoning ability learned through ARC would also work in more complex environments has not been revealed. Second, solving ARC does not imply the presence of other components of intelligence beyond reasoning. While reasoning is undoubtedly a core aspect of cognitive processes, it is not the entirety of intelligence. There is research showing that solving human-level complex tasks requires various cognitive abilities [20].

A.3.2 Will the Model Think Like Humans? Even if we assume that the ARC solver can reason in terms of LoTH, we cannot guarantee whether this solver's process is human-like for the following two reasons. Firstly, the current ARC provides a performance measure that rewards only for solving a task. It is important to recognize that such a measure might instigate the wrong purpose, leading to what is known as the King Midas problem [50]. This problem emphasizes the risk of AI achieving its given objective too literally, leading to unintended negative consequences, underscoring the importance of aligning AI's goals with human values and the broader context. The policy of rewarding only the results, excluding the solution process, makes it difficult to evaluate whether the solution process is similar to human reasoning. Therefore, models trained on current ARC likely differ in how they solve tasks compared to humans. The second reason is that directly comparing the reasoning processes of humans and language models is challenging. The process by which humans solve ARC tasks has not been investigated, making it unclear how the solving process differs between humans and artificial intelligence. Furthermore, there is a lack of metrics for comparing the solving processes, making direct comparisons difficult.

B Experimental Detail

B.1 Logical Coherence

The logical coherence study comprised two main experiments: a comparison on semantic coherence across prompting techniques and an assessment of the inferential coherence of LLMs. For the first experiment, prompting technique comparison, we randomly selected 100 tasks from the ARC evaluation set. We then applied three different prompting methods - Chain of Thought (CoT), Least to Most (LtM), and Tree of Thoughts (ToT) - to compare their effectiveness in maintaining semantic coherence.

The second experiment assessing the inferential coherence of the LLMs aims to assess whether the same logic can be consistently applied. Therefore, it is necessary to first confirm the tasks where the LLMs have understood the logic. To this end, we experimented using CoT prompting, which showed the best performance in the comparison across prompting techniques experiment, to solve the ARC training set. This experiment was repeated five times. The inferential coherence of the LLMs experiment was then conducted on tasks that were correctly solved at least once out of the five repetitions. The detailed task IDs and prompts used in each experiment are provided in B.1.1 and B.1.2, respectively.

B.1.1 Task ID List for Each Experiment. Task ID list selected for the experiment comparing logical coherence. The first experiment for comparison across prompting techniques was conducted on 100 ARC evaluation tasks, while the second experiment for the inferential coherence experiment of LLMs was carried out on 83 ARC training tasks.

Task ID List: Comparison Across Prompting Techniques

‘3ed85e70’, ‘0f63c0b9’, ‘17cae0c1’, ‘47996f11’, ‘4acc7107’, ‘0692e18c’, ‘477d2879’, ‘1c0d0a4b’, ‘292dd178’, ‘1990f7a8’, ‘22a4bbc2’, ‘4364c1c4’, ‘2f0c5170’, ‘17b80ad2’, ‘03560426’, ‘0c786b71’, ‘3391f8c0’, ‘42a15761’, ‘0bb8deee’, ‘1e97544e’, ‘1c02dbbe’, ‘4b6b68e5’, ‘2a5f8217’, ‘3194b014’, ‘1acc24af’, ‘0c9aba6e’, ‘0e671a1a’, ‘37d3e8b2’, ‘0becf7df’, ‘0607ce86’, ‘3a301edc’, ‘2546ccf6’, ‘009d5c81’, ‘31adaf00’, ‘281123b4’, ‘3d31c5b3’, ‘423a55dc’, ‘1d0a4b61’, ‘1a2e2828’, ‘319f2597’, ‘3979b1a8’, ‘12422b43’, ‘140c817e’, ‘0a2355a6’, ‘19bb5feb’, ‘332efdb3’, ‘27a77e38’, ‘2c0b0aff’, ‘00dbd492’, ‘2c737e3’, ‘2072aba6’, ‘48f8583b’, ‘27f8ce4f’, ‘14754a24’, ‘32e9702f’, ‘195ba7dc’, ‘137f0df0’, ‘184a9768’, ‘29700607’, ‘1c56ad9f’, ‘15663ba9’, ‘4c177718’, ‘136b0064’, ‘0a1d4ef5’, ‘1d398264’, ‘09c534e7’, ‘2685904e’, ‘48131b3c’, ‘31d5ba1a’, ‘2697da3f’, ‘103eff5b’, ‘12997ef3’, ‘1e81d6f9’, ‘25094a63’, ‘08573cc6’, ‘20981f0e’, ‘4852f2fa’, ‘2b01abd0’, ‘2072aba6’, ‘1a6449f1’, ‘34b99a2b’, ‘0b17323b’, ‘15696249’, ‘414297c0’, ‘2753e76c’, ‘12eac192’, ‘0934a4d8’, ‘310f3251’, ‘358ba94e’, ‘21f83797’, ‘4aab4007’, ‘351d6448’, ‘45bbe264’, ‘456873bc’, ‘15113be4’, ‘3490cc26’, ‘3b4c2228’, ‘00576224’, ‘42918530’, ‘45737921’, ‘20818e16’

Task ID List: Inferential Coherence of LLMs

‘017c7c7b’, ‘025d127b’, ‘08ed6ac7’, ‘0dfd9992’, ‘1bfc4729’, ‘22eb0ac0’, ‘239be575’, ‘23b5c85d’, ‘25d8a9c8’, ‘25ff71a9’, ‘272f95fa’, ‘27a28665’, ‘3618c87e’, ‘3af2c5a8’, ‘3bd67248’, ‘3bdb4ada’, ‘44f52bb0’, ‘48d8fb45’, ‘496994bd’, ‘49d1d64f’, ‘539a4f51’, ‘53b68214’, ‘5582e5ca’, ‘5bd6f4ac’, ‘6150a2bd’, ‘62c24649’, ‘67a3c6ac’, ‘67e8384a’, ‘68b16354’, ‘6d0aefbc’, ‘6f8cd79b’, ‘6fa7a44f’, ‘7447852a’, ‘746b3537’, ‘74dd1130’, ‘7837ac64’, ‘794b24be’, ‘7b7f7511’, ‘7f4411dc’, ‘82819916’, ‘88a62173’, ‘8be77c9e’, ‘8e1813be’, ‘90c28cc7’, ‘9172f3a0’, ‘963e52fc’, ‘97999447’, ‘9dfd6313’, ‘a416b8f3’, ‘a65b410d’, ‘a699fb00’, ‘a85d4709’, ‘a87f7484’, ‘aabf363d’, ‘b1948b0a’, ‘b8cdaf2b’, ‘b94a9452’, ‘ba26e723’, ‘ba97ae07’, ‘bc1d5164’, ‘bd4472b8’, ‘bda2d7a6’, ‘bdad9b1f’, ‘c3f564a4’, ‘c59eb873’, ‘c8f0f002’, ‘c9e6f938’, ‘c9f8e694’, ‘d0f5fe59’, ‘d10ecb37’, ‘d13f3404’, ‘d2abd087’, ‘d4469b4b’, ‘d631b094’, ‘d8c310e9’, ‘d9fac9be’, ‘dc433765’, ‘e26a3af2’, ‘e9afcf9a’, ‘ea786f4a’, ‘ef135b50’, ‘f5b8619d’, ‘f76d97a5’

B.1.2 Prompting Setting. The prompts used in comparison across prompting techniques and inferential coherence of LLMs are CoT, L2M, and ToT. These are detailed in Section B.1.3. In the prompts, parts enclosed in curly brackets indicate where the corresponding type should be inserted. Demo examples and test input refer to the demonstration examples and test input of the test example provided in the task to be solved. For instance, if the type is CoT Prompt, it consists of the CoT one-shot example, the task's demonstration examples, and test input. Regardless of the prompting method, all prompts are given a one-shot example. CoT solves the task using only the one-shot example, the task's demonstration examples, and test input. On the other hand, LtM and ToT use a decomposing prompt to obtain instructions for solving the problem through a decomposing stage. For LtM, the step-by-step solving prompt is then used to sequentially execute the instructions obtained through decomposing. The previously executed instructions and the resulting changed grid are included in this process. For ToT, the decomposing prompt is used to create multiple instruction candidates, and the ToT decomposing vote prompt is used to have the LLM select the most promising instruction candidate. The selected instructions are then processed through the step-by-step solving prompt to generate multiple candidate results for each instruction. The ToT step-by-step solving vote prompt is then used to select the grid that best reflects the instruction. This process is carried out step-by-step for all instructions.

B.1.3 Detailed Promptings. The logical coherence experiments employed various prompting techniques, including CoT, LtM, and ToT. CoT utilizes the CoT prompt, while LtM uses decomposing and step-by-step solving prompts. ToT incorporates decomposing, ToT decomposing vote, step-by-step solving, and ToT step-by-step solving vote prompts.

CoT One-Shot Example Data

If input grids are like that:
[[0, 3, 0, 0, 0, 0],
[0, 3, 0, 2, 0, 0],
[0, 0, 0, 2, 0, 0],
[0, 8, 0, 0, 2, 2],
[0, 0, 0, 0, 2, 2],
[6, 6, 6, 0, 0, 0]],

then these grids change to output grids below:
[[0, 0, 0, 0, 3, 0],
[0, 0, 0, 0, 3, 2],
[0, 0, 0, 0, 0, 2],
[0, 0, 0, 8, 2, 2],
[0, 0, 0, 0, 2, 2],
[0, 0, 0, 6, 6, 6]].

CoT One-Shot Example

Do you know the ARC problem?

It is similar to below.

{CoT One-Shot Example Data}

You can understand the pattern of this problem with the input-output pair in Example 1. In the above case, you can infer as follows.

In Example 1, all objects move to the right, and then you can move the object to the right side.

Like this concept, ‘object’, ‘count’, ‘color’, ‘move’, ‘row’, ‘column’, etc., help you to understand the patterns of the problem and solve it.

Below is another pattern to solve. So you can understand the pattern with several examples and apply the problem's input to get the correct output.

CoT Prompt

{CoT One-Shot Example}

{Demo Examples}

If input grids are like that:

{Test Input}

then output grids?

Decomposing One-Shot Example-Demo Examples

Example 1

If input grids are like that:
[[0, 0, 0, 0, 0, 0],
[0, 0, 3, 0, 0, 0],
[0, 3, 0, 3, 0, 0],
[0, 0, 3, 0, 3, 0],
[0, 0, 0, 3, 0, 0],
[0, 0, 0, 0, 0, 0]]

then these grids change to output grids below:
[[0, 0, 0, 0, 0, 0],
[0, 0, 3, 0, 0, 0],
[0, 3, 4, 3, 0, 0],
[0, 0, 3, 4, 3, 0],
[0, 0, 0, 3, 0, 0],
[0, 0, 0, 0, 0, 0]].

Example 2

If input grids are like that:
[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 3, 0, 3, 0, 0, 0, 0, 0],
[0, 0, 0, 3, 0, 3, 0, 0, 0, 0],
[0, 0, 3, 0, 0, 0, 3, 0, 0, 0],
[0, 0, 0, 0, 0, 3, 0, 3, 0, 0],
[0, 0, 0, 3, 0, 3, 3, 0, 0, 0],
[0, 0, 3, 3, 3, 0, 0, 0, 0, 0],
[0, 0, 0, 3, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]

then these grids change to output grids below:
[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 3, 0, 3, 0, 0, 0, 0, 0],
[0, 0, 0, 3, 0, 3, 0, 0, 0, 0],
[0, 0, 3, 0, 0, 0, 3, 0, 0, 0],
[0, 0, 0, 0, 0, 3, 4, 3, 0, 0],
[0, 0, 0, 3, 0, 3, 3, 0, 0, 0],
[0, 0, 3, 3, 3, 0, 0, 0, 0, 0],
[0, 0, 0, 3, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]].

Decomposing One-Shot Example-Test Input

Example-problem

If input grids are like that:
[[0, 0, 0, 0, 0, 3, 0, 0, 0, 0],
[0, 0, 0, 0, 3, 0, 0, 0, 0, 0],
[0, 3, 3, 0, 3, 3, 0, 3, 0, 0],
[3, 0, 0, 3, 0, 0, 3, 0, 3, 0],
[0, 0, 0, 3, 0, 0, 3, 3, 0, 0],
[0, 0, 0, 3, 0, 0, 3, 0, 0, 0],
[0, 0, 0, 3, 0, 0, 3, 0, 0, 0],
[0, 0, 0, 0, 3, 3, 0, 3, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 3, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]

then output grids?

Decomposing Prompt

Do you know the ARC problem?

Each example has the same pattern and the quiz also has the same pattern with examples. So if you understand the pattern of examples, you can solve the quiz. It will help you to analyze the pattern if you decompose the pattern into some steps. I give an example that decomposes patterns into subquestions.

{Decomposing One-Shot Example-Demo Examples}

{Decomposing One-Shot Example-Test Input}

To solve the quiz, I think we should do something like below:

Q1: We need to identify the places surrounded by 3s in the input grid of example-quiz.
Q2: Fill in the 4 in the location we found through Q1.

{Demo Examples}

{Test Input}

I want you to answer in the format below:

Q1: ....
Q2: ....
...
QN: ....

N is the index of the last question.

(The answers to the last question should allow you to generate the output grid for the quiz, and you shouldn't solve the problem yet in this process. You should only create the subquestions for solving the problem.)

Step-By-Step Solving Prompt

{CoT One-Shot Example}

{Demo Examples}

If input grids are like that:
{Test Input}

then output grids?

{Previous Instructions}

{Previous Changed Grid}

{Current Instruction}

ToT Decomposing Vote Prompt

First, consider how to solve the problem below.

{Demo Examples}

{Test Input}

Then, given instruction and several choices, decide which choice is most promising. Analyze each choice in detail, then conclude in the last line "The best choice is s", where s is the integer ID of the choice.

ToT Step-By-Step Solving Vote Prompt

First, consider how to solve the problem below.

{Demo Examples}

{Test Input}

Then, given a question and some answers, you need to choose which answer is the best answer to the question. Analyze each choice in detail and then conclude on the last line, "The best answer is 's'," where s is the integer ID of the answer.

{Previous Instructions}

{Previous Changed Grid}

{Current Instruction}

{Current Changed Grid}

B.2 Compositionality

In the compositionality study, we conducted two experiments: the LLMs DSL understanding experiment to assess how well LLMs comprehend the provided DSLs, and an experiment to evaluate LLMs' compositionality ability. The LLMs DSL understanding experiment measures how accurately LLMs can generate the correct DSLs when given the answer for a task. The compositionality ability experiment examines whether LLMs can correctly select and use the necessary DSLs from those provided for problem-solving. Both experiments used the same set of tasks. Detailed information about the task IDs can be found in Table B.2.1, while the specific prompt details are available in Table B.2.4 and Table B.2.6.

B.2.1 Task ID List. Task ID list for the compositionality experiment comprising 158 tasks. From the total 800 ARC tasks, we selected only those problems where the input and output grid sizes were identical and could be solved with DSL sequence length within 10 using the given DSL for our experiment.

Task ID List: LLMs DSL Understanding & Compositionality of LLMs

'025d127b', '05f2a901', '08ed6ac7', '0ca9ddb6', '0d3d703e', '11852cab', '150deff5', '1b60fb0c', '1caeab9d', '1e0a9b12', '1f642eb9', '1f876c06', '2204b7a8', '22168020', '22233c11', '228f6490', '22eb0ac0', '253bf280', '25d487eb', '25d8a9c8', '25ff71a9', '29c11459', '2bee17df', '2c608aff', '2dd70a9a', '3345333e', '3618c87e', '36fdfd69', '3906de3d', '3aa6fb7a', '3bd67248', '3c9b0459', '3e980e27', '3eda0437', '40853293', '42a50994', '444801d8', '44d8ac46', '4938f0c2', '496994bd', '50846271', '508bd3b6', '50cb2852', '5168d44c', '54d82841', '5582e5ca', '56dc2b01', '56ff96f3', '5c0a986e', '60b61512', '6150a2bd', '623ea044', '63613498', '67385a82', '673ef223', '67a3c6ac', '67a423a3', '6855a6e4', '68b16354', '694f12f3', '6c434453', '6cf79266', '6d58a25d', '6d75e8bb', '6e02f1e3', '6e19193c', '6e82a1ae', '74dd1130', '760b3cac', '776ffc46', '794b24be', '7ddcd7ec', '7e0986d6', '7f4411dc', '810b9b61', '855e0971', '88a10436', '890034e9', '8d510a79', '90f3ed37', '928ad970', '93b581b8', '941d9a10', '952a094c', '9565186b', '99fa7670', '9dfd6313', 'a2fd1cf0', 'a3df8b1e', 'a48eeaf7', 'a5313dff', 'a61f2674', 'a65b410d', 'a699fb00', 'a78176bb', 'a79310a0', 'a85d4709', 'a9f96cdd', 'aabf363d', 'ae3edfdc', 'aedd82e4', 'af902bf9', 'b1948b0a', 'b230c067', 'b2862040', 'b548a754', 'b7249182', 'b8cdaf2b', 'ba97ae07', 'bb43febb', 'bda2d7a6', 'bdad9b1f', 'c0f76784', 'c8f0f002', 'cbded52d', 'ce22a75a', 'ce9e57f2', 'd037b0a7', 'd07ae81c', 'd23f8c26', 'd2abd087', 'd406998b', 'd43fd935', 'd4a91cb9', 'd4f3cd78', 'd511f180', 'd5d6de2d', 'd6ad076f', 'd89b689b', 'd8c310e9', 'd90796e8', 'd9f24cd1', 'dc433765', 'ddf7fa4f', 'ded97339', 'e40b9e2f', 'e48d4e1a', 'e5062a87', 'e509e548', 'e73095fd', 'e8dc4411', 'e9614598', 'e9afcf9a', 'ea32f347', 'ea786f4a', 'ec883f72', 'ed36ccf7', 'ef135b50', 'f25ffba3', 'f76d97a5', 'f8a8fe49', 'fcc82909', '8f2ea7aa', '5521c0d9', '32597951', '98cf29f8', '0e206a2e', 'a1570a43'

B.2.2 Types of DSLs used. Each DSL was implemented as a Python function. As shown in Table 7, there are three types of DSLs using three parameter types. Color Change DSLs accept parameters such as Coordinate and Object. Coordinate-based Color Change DSLs include Pixel Color, X Line, Horizontal Line, Vertical Line, and Diagonal Line. For Object parameters, only the obj color DSL exists. Transformation DSLs use Object and Grid parameters. Object-based transformations include Rotate Left Obj, Rotate Right Obj, Horizontal Flip Obj, Vertical Flip Obj, and movement operations (Move Left, Move Right, Move Up, Move Down). Grid-based transformations include Rotate Left State, Rotate Right State, Horizontal Flip, and Vertical Flip. Lastly, the Complete DSL exists independently of parameters, indicating task completion before reaching DSL sequence length of 10. For tasks solved with exactly DSL sequence length of 10, the Complete DSL is unnecessary.

Table 7. DSL list: The DSL used in the compositionality experiment is categorized based on the type of target and the kind of functionality. The targets in the DSL are categorized into three types: Coordinate, Object, and Grid. The functions of the DSL include changing the color of a target (Color Change), moving a target (Transformation), and indicating the completion of the task at DSL sequence length shorter than 10 (Complete).

	Coordinate	**Object**	**Grid**
Color Change	Pixel Color, X Line, Horizontal Line, Vertical Line, Diagonal Line	Obj Color	X
Transformation	X	Rotate Left Obj, Rotate Right Obj, Horizontal Flip Obj, Vertical Flip Obj, Move Left, Move Right, Move Up, Move Down	Rotate Left State, Rotate Right State, Horizontal Flip, Vertical Flip
Complete		Complete	

B.2.3 Prompt Contents for LLMs with DSL Codes and Comments . In the two experiments measuring compositionality and LLM's DSL understanding, we identified a set of 10 tasks that collectively required the use of all 15 DSLs at least once. This set was used to determine the optimal prompt for explaining DSLs to LLMs. We conducted experiments with four prompt variants: no DSL information, DSL code only, DSL comments only, and both DSL code and comments. The LLMs DSL understanding experiment was performed for these 10 tasks across all four prompt compositions. Results indicated that providing both code and comments yielded optimal performance. Consequently, we employed prompts containing both DSL code and comments for the LLM's DSL understanding and compositionality of LLMs experiments. Section B.2.4 illustrates the prompt content where both code and comments were provided to the LLM.

B.2.4 *Detailed DSL Promptings.* Prompts of DSL Function Codes and Comments

DSL Prompt: Rotate Left State

```
# rotate_left_state function is a counterclockwise rotation about the given state.
# This function rotates a square grid (NxN) counterclockwise by 90 degrees.
# Parameters:
# - state: A 2D list representing the current grid state.
# Returns:
# - A new 2D list representing the grid after the counterclockwise rotation.

def rotate_left_state(state):
    N = len(state)
    rotated_state = copy.deepcopy(state)
    if N == len(state[0]):
        temp_state = copy.deepcopy(state)
        for x in range(N):
            for y in range(N):
                rotated_state[N−1−y][x] = state[x][y]
    return rotated_state
```

DSL Prompt: Rotate Right State

```
# rotate_right_state function is a clockwise rotation about the given state.
# This function rotates a square grid (NxN) clockwise by 90 degrees.
# Parameters:
# - state: A 2D list representing the current grid state.
# Returns:
# - A new 2D list representing the grid after the clockwise rotation.

def rotate_right_state(state):
    N = len(state)
    rotated_state = copy.deepcopy(state)
    if N == len(state[0]):
        for x in range(N):
            for y in range(N):
                rotated_state[y][N−1−x] = state[x][y]
    return rotated_state
```

DSL Prompt: Vertical Flip

```
# vertical_flip function is a flip by x-axis about the given state.
# This function flips the grid vertically, swapping the top and bottom rows.
# Parameters:
```

```
# - state: A 2D list representing the current grid state.
# Returns:
# - A new 2D list representing the grid after the vertical flip.

def vertical_flip(state):
    temp_state = copy.deepcopy(state)
    N = len(state)
    M = len(state[0])
    for i in range(N):
        for j in range(M):
            temp_state[N−1−i][j] = state[i][j]
    return temp_state
```

DSL Prompt: Horizontal Flip

```
# horizontal_flip function is a flip by y-axis about the given state.
# This function flips the grid horizontally, swapping the left and right columns.
# Parameters:
# - state: A 2D list representing the current grid state.
# Returns:
# - A new 2D list representing the grid after the horizontal flip.

def horizontal_flip(state):
    N = len(state)
    M = len(state[0])
    flipped_state = copy.deepcopy(state)
    for i in range(N):
        for j in range(M // 2):
            flipped_state[i][j], flipped_state[i][M−1−j] = state[i][M−1−j], state[i][j]
    return flipped_state
```

DSL Prompt: Move Right

```
# move_right function moves all pixels in the selected object to the right by one column.
# Parameters:
# - state: A 2D list representing the current grid state.
# - object: A list of lists where each inner list contains the coordinates [x, y] of a pixel to
move.
# Returns:
# - A new 2D list representing the grid after the object is moved to the right.

def move_right(state, object):
    move_state = copy.deepcopy(state)
    new_obj = []
    for x, y in object:
        move_state[x][y] = 0
    for x, y in object:
        new_x, new_y = x, y + 1
        if 0 <= new_x < len(state) and 0 <= new_y < len(state[0]):
            move_state[new_x][new_y] = state[x][y]
            new_obj.append([new_x, new_y])
    return move_state
```

DSL Prompt: Move Left

```
# move_left function moves all pixels in the selected object to the left by one column.
# Parameters:
# - state: A 2D list representing the current grid state.
# - object: A list of lists where each inner list contains the coordinates [x, y] of a pixel to
move.
# Returns:
# - A new 2D list representing the grid after the object is moved to the left.

def move_left(state, object):
    move_state = copy.deepcopy(state)
    new_obj = []
    for x, y in object:
        move_state[x][y] = 0
    for x, y in object:
        new_x, new_y = x, y − 1
        if 0 <= new_x < len(state) and 0 <= new_y < len(state[0]):
            move_state[new_x][new_y] = state[x][y]
            new_obj.append([new_x, new_y])
    return move_state
```

DSL Prompt: Move Up

```
# move_up function moves all pixels in the selected object up by one row.
# Parameters:
# - state: A 2D list representing the current grid state.
# - object: A list of lists where each inner list contains the coordinates [x, y] of a pixel to
move.
# Returns:
# - A new 2D list representing the grid after the object is moved up.
```

```
def move_up(state, object):
    move_state = copy.deepcopy(state)
    new_obj = []
    for x, y in object:
        move_state[x][y] = 0
    for x, y in object:
        new_x, new_y = x−1, y
        if 0 <= new_x < len(state) and 0 <= new_y < len(state[0]):
            move_state[new_x][new_y] = state[x][y]
            new_obj.append([new_x, new_y])
    return move_state
```

DSL Prompt: Move Down

```
# move_down function moves all pixels in the selected object down by one row.
# Parameters:
# - state: A 2D list representing the current grid state.
# - object: A list of lists where each inner list contains the coordinates [x, y] of a pixel to
move.
# Returns:
# - A new 2D list representing the grid after the object is moved down.

def move_down(state, object):
    move_state = copy.deepcopy(state)
    new_obj = []
    for x, y in object:
        move_state[x][y] = 0
    for x, y in object:
        new_x, new_y = x + 1, y
        if 0 <= new_x < len(state) and 0 <= new_y < len(state[0]):
            move_state[new_x][new_y] = state[x][y]
            new_obj.append([new_x, new_y])
    return move_state
```

DSL Prompt: Rotate Right Object

```
# rotate_right_obj function makes a clockwise rotation about the given object.
# This function rotates the selected object within the grid 90 degrees clockwise around its
center.
# Parameters:
# - state: A 2D list representing the current grid state.
# - object: A list of lists where each inner list contains the coordinates [x, y] of a pixel in
the object.
# Returns:
# - A new 2D list representing the grid after the object is rotated clockwise.

def rotate_right_obj(state, object):
    rotate_state = copy.deepcopy(state)
    new_obj = []
    max_x = max(x for x,_ in object)
    min_x = min(x for x,_ in object)
    max_y = max(y for_, y in object)
    min_y = min(y for_, y in object)
    fixed_x = (max_x + min_x) // 2
    fixed_y = (max_y + min_y) // 2

    for x, y in object:
        rotate_state[x][y] = 0

    for x, y in object:
        moved_x = y − fixed_y + fixed_x
        moved_y = −x + fixed_x + fixed_y
        if 0 <= moved_x < len(state) and 0 <= moved_y < len(state[0]):
            rotate_state[moved_x][moved_y] = state[x][y]
            new_obj.append([moved_x, moved_y])

    return rotate_state
```

DSL Prompt: Rotate Left Object

```
# rotate_left_obj function makes a counterclockwise rotation about the given object.
# This function rotates the selected object within the grid 90 degrees counterclockwise
around its center.
# Parameters:
# - state: A 2D list representing the current grid state.
# - object: A list of lists where each inner list contains the coordinates [x, y] of a pixel in
the object.
# Returns:
# - A new 2D list representing the grid after the object is rotated counterclockwise.

def rotate_left_obj(state, object):
    rotate_state = copy.deepcopy(state)
    new_obj = []
    max_x = max(x for x,_ in object)
    min_x = min(x for x,_ in object)
    max_y = max(y for_, y in object)
    min_y = min(y for_, y in object)
    fixed_x = (max_x + min_x) // 2
    fixed_y = (max_y + min_y) // 2

    for x, y in object:
        rotate_state[x][y] = 0

    for x, y in object:
        moved_x = −y + fixed_y + fixed_x
        moved_y = x − fixed_x + fixed_y
        if 0 <= moved_x < len(state) and 0 <= moved_y < len(state[0]):
            rotate_state[moved_x][moved_y] = state[x][y]
            new_obj.append([moved_x, moved_y])

    return rotate_state
```

DSL Prompt: Vertical Flip Object

```
# vertical_flip_obj function makes a vertical flip of the selected object.
# This function flips the selected object within the grid vertically.
# Parameters:
# - state: A 2D list representing the current grid state.
# - object: A list of lists where each inner list contains the coordinates [x, y] of a pixel in
the object.
# Returns:
# - A new 2D list representing the grid after the object is flipped vertically.

def vertical_flip_obj(state, object):
    flip_state = copy.deepcopy(state)
    new_obj = []
    max_x = max(x for x,_ in object)
    min_x = min(x for x,_ in object)

    for x, y in object:
        flip_state[x][y] = 0

    for x, y in object:
        flip_state[max_x + min_x − x][y] = state[x][y]
        new_obj.append([max_x + min_x − x, y])
    return flip_state
```

DSL Prompt: Horizontal Flip Object

```
# horizontal_flip_obj function makes a horizontal flip of the selected object.
# This function flips the selected object within the grid horizontally.
# Parameters:
# - state: A 2D list representing the current grid state.
# - object: A list of lists where each inner list contains the coordinates [x, y] of a pixel in
the object.
# Returns:
# - A new 2D list representing the grid after the object is flipped horizontally.

def horizontal_flip_obj(state, object):
    flip_state = copy.deepcopy(state)
    new_obj = []
    max_y = max(y for_, y in object)
    min_y = min(y for_, y in object)

    for x, y in object:
        flip_state[x][y] = 0

    for x, y in object:
        flip_state[x][max_y + min_y − y] = state[x][y]
        new_obj.append([x, max_y + min_y − y])

    return flip_state
```

DSL Prompt: X Line

```
# X_line function makes a diagonal X-line in all directions from a given pixel until they
reach the end of the grid.
# Parameters:
# - state: A 2D list representing the current grid state.
# - r: The row index of the starting pixel.
# - c: The column index of the starting pixel.
# - color: The color to be used for the X-line.
# Returns:
# - A new 2D list representing the grid after the X-line is drawn.

def X_line(state, r, c, color):
    X_state = copy.deepcopy(state)
    x_move = [−1, 1]
    y_move = [−1, 1]

    for i in x_move:
        for j in y_move:
            moved_x, moved_y = r + i, c + j
            while 0 <= moved_x < len(state) and 0 <= moved_y < len(state[0]):
                X_state[moved_x][moved_y] = color
                moved_x += i
                moved_y += j

    return X_state
```

DSL Prompt: Horizontal Line

```
# horizontal_line function draws a horizontal line between two pixels if they are on the
same row.
# Parameters:
# - state: A 2D list representing the current grid state.
# - r1: The row index of the first pixel.
# - c1: The column index of the first pixel.
# - r2: The row index of the second pixel.
# - c2: The column index of the second pixel.
# - color: The color to be used for the line.
# Returns:
# - A new 2D list representing the grid after the horizontal line is drawn.

def horizontal_line(state, r1, c1, r2, c2, color):
    line_state = copy.deepcopy(state)
    if r1 == r2:
        if c1 < c2:
            if c2 < len(state[0]):
                for i in range(c1+1, c2):
                    line_state[r1][i] = color
        else:
            if c1 < len(state[0]):
                for i in range(c2+1, c1):
                    line_state[r1][i] = color
    return line_state
```

DSL Prompt: Vertical Line

```
# vertical_line function draws a vertical line between two pixels if they are in the same column.
# Parameters:
# - state: A 2D list representing the current grid state.
# - r1: The row index of the first pixel.
# - c1: The column index of the first pixel.
# - r2: The row index of the second pixel.
# - c2: The column index of the second pixel.
# - color: The color to be used for the line.
# Returns:
# - A new 2D list representing the grid after the vertical line is drawn.

def vertical_line(state, r1, c1, r2, c2, color):
    line_state = copy.deepcopy(state)
    if c1 == c2:
        if r1 < r2:
            if r2 < len(state):
                for i in range(r1+1, r2):
                    line_state[i][c1] = color
        else:
            if r1 < len(state):
                for i in range(r2+1, r1):
                    line_state[i][c1] = color
    return line_state
```

DSL Prompt: Diagonal Line

```
# diagonal_line function draws a diagonal line between two pixels if they form a 45-degree angle.
# Parameters:
# - state: A 2D list representing the current grid state.
# - r1: The row index of the first pixel.
# - c1: The column index of the first pixel.
# - r2: The row index of the second pixel.
# - c2: The column index of the second pixel.
# - color: The color to be used for the line.
# Returns:
# - A new 2D list representing the grid after the diagonal line is drawn.

def diagonal_line(state, r1, c1, r2, c2, color):
    line_state = copy.deepcopy(state)
    if abs(r1 − r2) == abs(c1 − c2):
        dr = 1 if r2 > r1 else −1
        dc = 1 if c2 > c1 else −1
        r, c = r1 + dr, c1 + dc
        while r != r2 and c != c2:
            line_state[r][c] = color
            r += dr
            c += dc
    return line_state
```

DSL Prompt: Object Color

```
# obj_color function changes the color of the selected object.
# Parameters:
# - state: A 2D list representing the current grid state.
# - object: A list of lists where each inner list contains the coordinates [x, y] of a pixel in
the object.
# - color: The new color to be applied to the object.
# Returns:
# - A new 2D list representing the grid after the object's color is changed.

def obj_color(state, object, color):
    color_state = copy.deepcopy(state)
    for x, y in object:
        color_state[x][y] = color
    return color_state
```

DSL Prompt: Pixel Color

```
# pixel_color function changes the color of the selected pixel.
# Parameters:
# - state: A 2D list representing the current grid state.
# - r: The row index of the pixel to change.
# - c: The column index of the pixel to change.
# - color: The new color to be applied to the pixel.
# Returns:
# - A new 2D list representing the grid after the pixel's color is changed.

def pixel_color(state, r, c, color):
    temp_state = copy.deepcopy(state)
    temp_state[r][c] = color
    return temp_state
```

DSL Prompt: Complete

```
# complete function returns the current state as the final answer of the quiz.
# Parameters:
# - state: A 2D list representing the current grid state.
# Returns:
# - The same 2D list representing the final grid state.

def complete(state):
    return state
```

B.2.5 Prompt of Compositionality Experiment. Both the LLMs DSL understanding and compositionality of LLMs experiments utilized the prompt structure outlined in Section B.2.6. The Introduction ARC Prompt provides a comprehensive overview of ARC, while the DSL Usage Example Prompt illustrates DSL application. The DSL prompt, comprising the prompts of DSL function codes and comments from Section B.2.4 and the DSL usage example prompt, offers a comprehensive DSL explanation. The task prompt includes demonstration examples, test input, object information (coordinates of objects obtained through PnP in dictionary format), and output format guidelines. In the case of the LLMs DSL understanding prompt, unlike the task prompt, the DSLs path for the task is provided. The CoT prompt included the introduction ARC prompt and DSL prompt. In the case of the LLMs DSL understanding experiment, the LLMs DSL understanding prompt was used, while in the case of the compositionality of LLMs experiment, the task prompt was used. In the compositionality experiments, the CoT prompt was utilized.

B.2.6 Detailed Promptings. Composition of prompt contents used in the compositionality experiments.

Introduction ARC Prompt

Do you know ARC problem?

ARC is a quiz, and if we can solve this problem, we can understand and utilize several concepts such as “object", “count", “color", “move", “row", “column", etc.

ARC problems give you some examples to understand these patterns. You can understand the pattern below with several examples and then apply the quiz’s input to get the correct output.

DSL Usage Example Prompt

In this example grid,
[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 5, 5, 0],
[0, 5, 5, 0, 0, 0, 0, 5, 5, 0],
[0, 0, 5, 5, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 5],

```
[0, 0, 0, 0, 0, 5, 5, 0, 0, 5],
[0, 5, 0, 0, 0, 0, 0, 0, 0, 5],
[0, 5, 0, 0, 5, 0, 0, 0, 0, 0],
[0, 0, 0, 5, 5, 0, 0, 0, 0, 0]]
```

there are 6 objects
Object1: [[1, 7], [1, 8], [2, 7], [2, 8]]
Object2: [[2, 1], [2, 2], [3, 2], [3, 3]]
Object3: [[5, 9], [6, 9], [7, 9]]
Object4: [[6, 5], [6, 6]]
Object5: [[7, 1], [8, 1]]
Object6: [[8, 4], [9, 3], [9, 4]]

If you apply “rotate_right_obj(state, object2)", the result becomes

```
[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 5, 0, 0, 0, 0, 5, 5, 0],
[0, 5, 5, 0, 0, 0, 0, 5, 5, 0],
[0, 5, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 5],
[0, 0, 0, 0, 0, 5, 5, 0, 0, 5],
[0, 5, 0, 0, 0, 0, 0, 0, 0, 5],
[0, 5, 0, 0, 5, 0, 0, 0, 0, 0],
[0, 0, 0, 5, 5, 0, 0, 0, 0, 0]]
```

DSL Prompt

{Prompts of DSL Function Codes and Comments}

Arguments for the DSLs are mainly “state" and “object", but some require “color", “row", “column", etc. “state" is the current state of the grid, which is the entire grid.

"object" is the list of coordinates of the object; there may be multiple objects in the grid, but no DSL requires multiple objects. **“color"** is the color of the pixel in the grid, which is a number between 0 and 9. **“row"** and **“column"** are the coordinate numbers of a pixel in the grid.

You can choose from here and apply the DSL to solve the problem. You must input the appropriate arguments to the DSL, or it will not work.

{DSL Usage Example Prompt}

Please choose the DSL from the list above and provide the proper arguments to solve the problem.

Task Prompt

{Demo Examples} & **{Test Input}**

You must solve the given quiz within 10 steps! Select one DSL with proper arguments in each step.

If you think the current state is correct, you can select the “complete" DSL.

I want you to answer in the format below.

The output should be in the following JSON format:
{

‘step’: “(current_step)",

‘dsl’: “(dsl with the arguments for the DSL)",

‘description’: “(why you chose this DSL?)"
}

CoT Prompt

{Introduction ARC Prompt}

{DSL Prompt}

{Task Prompt}

B.3 Productivity

In the productivity experiment, we aimed to augment the task's demonstration example pairs using the Inverse Transformation Prompt (ITP). The ITP consists of a category prompt containing a description of the category, example pairs, and the target output to be augmented. The detailed structure of the category prompt is provided in Section B.3.2, and the structure of the ITP is outlined in Section B.3.1.

B.3.1 ITP. Composition of prompt contents used in the productivity experiments. ITP consists of a category prompt, example pairs, and target output.

ITP

Try solving the ARC problem and do not say sensitive word: generate the output accordingly. I will give you a hint.

{Category Prompt}

Here are some examples to help you.

{Example Pairs}

{Target Output}

Provide a two-dimensional array that is not identical to the input array or a direct copy of the example. Each element is an integer between 0 and 9. Would you give me 2 answers? No need to explain how you solved it.

B.3.2 Category Prompts. These category prompts explain the 16 types of ConceptARC. In Section B.3.1, the category prompt is filled with the appropriate prompt corresponding to the category of the task to be generated.

Category Prompt: AboveBelow

Focus on the horizontal criteria, you may have to modify some regions by that line, such as removing, moving, filling region by color element. See the provided example to how to modify.

Category Prompt: Center

Fix the array issue by addressing the center, potentially moving or removing the central element. See the provided example for clarity.

Category Prompt: CleanUp

Distort the shapes in areas where they are polygonal or completely filled, adding noise or disturbances to disrupt the complete shapes.

Category Prompt: CompleteShape

Distort the perfectly shaped objects identified in the input image. Introduce noise to these identified objects to easily generate diverse outputs.

Category Prompt: Copy

Delete one identical object from the output. Refer to the example to identify which one to remove. Consider deleting the object located in a position-indicating space.

Category Prompt: Count

Create an input image based on the provided count-related problem. Focus on details like object or color count, as shown in the example.

Category Prompt: ExtendToBoundary

In the input image, find lines connected to boundaries with different colors. Transform these lines into a different shape. The example illustrates how to make this transformation.

Category Prompt: ExtractObjects

Generate an output image with objects from the given input. Refer to examples for guidance. Hint: Extract objects when inferring input from output.

Category Prompt: FilledNotFilled

When inferring the input from the output, focus on situations where the inner part of an object contains empty space or another object. Examples provide guidance for creating the output image.

Category Prompt: HorizontalVertical

Focus on horizontal and vertical relations, representing them with colors or preserving one direction while eliminating the other. Examples illustrate the approach.

Category Prompt: InsideOutside

Address the inside-outside relationship, either by selecting items inside or outside in the input or determining quantities. Use the boundary as a reference. Examples offer guidance.

Category Prompt: MoveToBoundary

Objects in the input may be shifted to one side, and in the output, they are displaced either horizontally or vertically. Infer the direction from examples and choose the displacement freely.

Category Prompt: Order

This is about randomly rearranging initially ordered objects while representing their original positions through a specific rule. Examine the examples to understand how to achieve this.

Category Prompt: SameDifferent

You'll notice that only specific-shaped objects are extracted in the input image. Create additional objects in the zero-represented space. Examples provide guidance on how to proceed.

Category Prompt: TopBottom2D

Objects are in a 2D space. Check changes in the top and bottom. The input may have shifted or require removing top/bottom indicators. Look at examples for specifics.