Evaluating Inductive Reasoning Capabilities of Large Language Models With The One Dimensional Abstract Reasoning Corpus

Cédric S. Mesnage[1,2], Xiaoyang Wang[2], Hang Dong[2] and Aishwaryaprajna[2]

[1]*Institute for Data Science and Artificial Intelligence, University of Exeter, Exeter, United Kingdom*
[2]*Department of Computer Science, University of Exeter, Exeter, United Kingdom*

Abstract
We present an initial automated test to evaluate LLMs' capacity to perform inductive reasoning tasks. We use the GPT-3.5 and GPT-4 models to create a system which generates Python code as hypotheses for inductive reasoning to transform sequences of the One Dimensional Abstract Reasoning Corpus (1D-ARC) challenge. We experiment with three prompting techniques, namely standard prompting, Chain of Thought (CoT), and direct feedback. We provide results and an analysis of cost-to-success rate and benefit-cost ratio. Our best result is an overall 25% success rate with our CoT prompting on GPT-4, significantly surpassing the standard prompting approach. We discuss potential avenues to improve our experiments and test other strategies, and combine deductive reasoning with LLM-based inductive reasoning.

Keywords
Large Language Models, Inductive Reasoning, Prompt Engineering, Abstract Reasoning Corpus

1. Introduction

Pre-trained large language models (LLMs) are approaching or have exceeded human-level performance in various tasks including abstractive summarization, question answering in some domains, some coding tasks, etc. [1, 2, 3]. Moving towards more intelligent systems, or Artificial General Intelligence (AGI) [4], there are significant interests in the reasoning abilities of LLMs, which is a fundamental aspect of human intelligence [5]. There are debates about whether LLMs can reason based on the understanding of truth and logic, which is closer to the "thinking" process of humans [6]. By leveraging *prompting* techniques (in-context few-shot learning), pre-trained LLMs can solve some reasoning tasks in different benchmarks, including math, commonsense, and games [7, 8], but their performance heavily depends on the prompting approach.

GPT-3 showcased the zero-shot inference ability, with limited ability to perform more complex reasoning tasks [9]. Few-shot learning is then studied to promote "thinking" by showcasing several examples of intermediate reasoning steps, named CoT [7]. Zero-shot CoT is proposed through simple additional prompting "*Let's think step-by-step*" [10]. While this research focuses on probing reasoning ability through in-context learning, we would like to further understand the variations of different prompting techniques. We would also like to understand the impact of feedback on the thinking process of LLMs, and how it would correct itself through feedback. Besides, there is a gap in benchmarking the cost of LLMs with different prompting techniques.

Inductive reasoning is a fundamental cognitive challenge that involves making generalizations from specific instances [11]. It requires inferring the principles from the observations and applying them to novel situations. Previous works on various types of reasoning tasks, including arithmetic, commonsense, and symbolic reasoning, have showcased the reasoning capabilities of LLMs to a certain degree [12, 13, 14, 15]. Given that LLMs are in-context few-shot learners [9], these studies on reasoning

3rd International Workshop on HYbrid Models for Coupling Deductive and Inductive ReAsoning (HYDRA 2024)
C.S.Mesnage@exeter.ac.uk (C. S. Mesnage); X.Wang7@exeter.ac.uk (X. Wang); H.Dong2@exeter.ac.uk (H. Dong); Aishwaryaprajna@exeter.ac.uk (Aishwaryaprajna)
0000-0002-2004-6378 (C. S. Mesnage); 0000-0001-9332-2700 (X. Wang); 0000-0001-6828-6891 (H. Dong); 0000-0003-4386-9745 (Aishwaryaprajna)

abilities have paved the way for novel prompting and searching techniques, such as the CoT [7] and Tree of Thoughts [8].

Inspired by the Bayesian learner, a hypothesis search method is proposed to assist LLMs in solving complex inductive reason tasks [16]. Instead of directly prompting, LLMs are used to generate natural language hypotheses according to the problem description and examples, then translate hypotheses into Python programs for execution and testing. This work has demonstrated the contribution of hypothesis searching and computer programs in solving complex inductive reasoning problems. Note that human-annotated hypotheses are introduced as few-shot demonstrations for generating new hypotheses. Human annotators are also involved in selecting the subset of hypotheses, to avoid the huge computation load in testing all hypotheses.

Existing work demonstrates the improvement of LLMs' reasoning abilities through thinking step-by-step or introducing hypotheses selection and testing loops. We propose to perform a comprehensive analysis of different prompting techniques, as well as different ways for LLMs to produce results, with minimal human intervention in the reasoning process. As the study [16] has demonstrated the contribution of computer programs in solving complex inductive reasoning problems, we also ask LLMs to generate computer programs for easy generalisation and testing. Thus, this task is relevant to program synthesis, a classical problem studied with inductive and deductive reasoning and more recently with neural networks and LLMs [17, 18]. We mainly test LLMs for inductive reasoning in this work and will discuss deductive reasoning at the end of the paper.

We are looking at evaluating the effect of CoT [7] and CoT variations [10] on solving reasoning tasks from the 1D-ARC [19] which we compare with standard prompting and direct feedback. We use the OpenAI API and store every interaction with it for each solving method that we set.

The goal of developing intelligent agents which would be capable of understanding the world by interpreting their observations requires theoretically formalising a model of the real world. Along this goal, we argue that observing a phenomenon, hypothesising and theorising about it requires the same skills as finding the transformation rules in the 1D-ARC tasks and that code is a way of formalising a theory. How good are LLMs at those skills? We evaluate the inductive reasoning capacity of LLMs as intelligent agents by automatically generating prompts to perform 1D-ARC tasks, prompting for code of transformation functions and testing the correctness of the generated functions. One path to building more capable LLMs is to provide them with specific solving methods such as CoT reasoning, multi-agent conversations or interactive systems. In this study, we focus on the effect of CoT and direct feedback compared to standard prompting. Since we are interested in finding a general approach we chose to evaluate the zero-shot CoT as it does not require the developer to produce an example of reasoning.

The remainder of the paper is as follows: Section 2 describes the 1D-ARC, prompting techniques, testing and evaluation metrics; Section 3 summarises the results; and Section 4 and Section 5 discuss the results and conclude the work with potential future research.

2. Methodology

This section describes the methods we developed to evaluate and compare different prompting methods. In order to evaluate the level of reasoning of current large language models, we develop a method to automatically assess the correctness of answers produced by an LLM. We choose the 1D-ARC as a dataset of tasks to perform since it provides us with a clear test to evaluate.

2.1. Abstraction and Reasoning Corpus

The 1D-ARC consists of 900 tasks grouped into 18 categories of 50 tasks each. For each task, we have 3 examples of input-to-output sequences to illustrate the transformation to be performed and one test set of one input and one output. Figure 1 is a graphical representation of each category. These are tasks which are simple to complete for humans. The sequences are represented as colored squares, the background ones are black and the colors of interest as other colors. For instance, a transformation

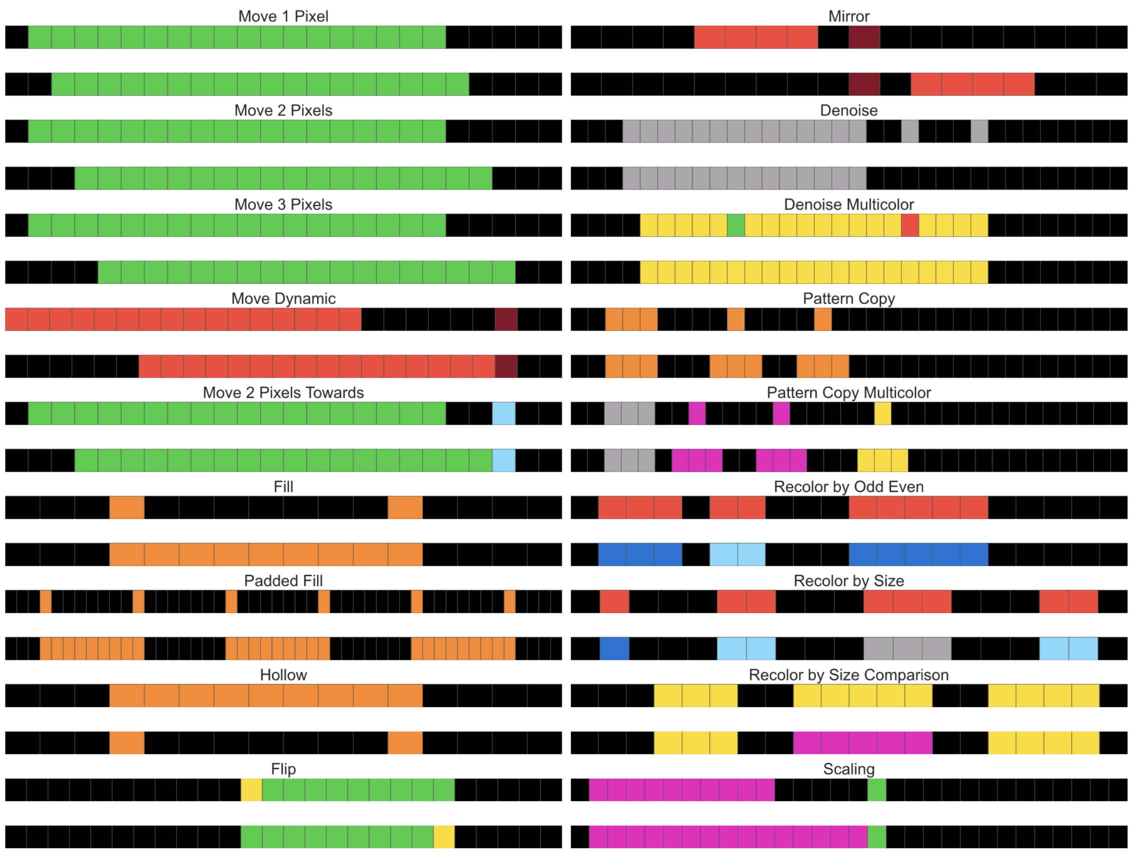


Figure 1: The 1D-ARC task categories and representative input-output pairs [19].

can be to shift all colored squares by one pixel to the right whilst keeping the black pixels identical. Another transformation is the mirror, which transforms the sequence into its reverse.

To evaluate the capacity of LLMs to understand those transformations, we generate prompts sent to the OpenAI application programming interface (API) for GPT-3.5/4 models for each of the tasks and ask for Python code to be generated. We describe this process in the following subsections.

2.2. Code generation process

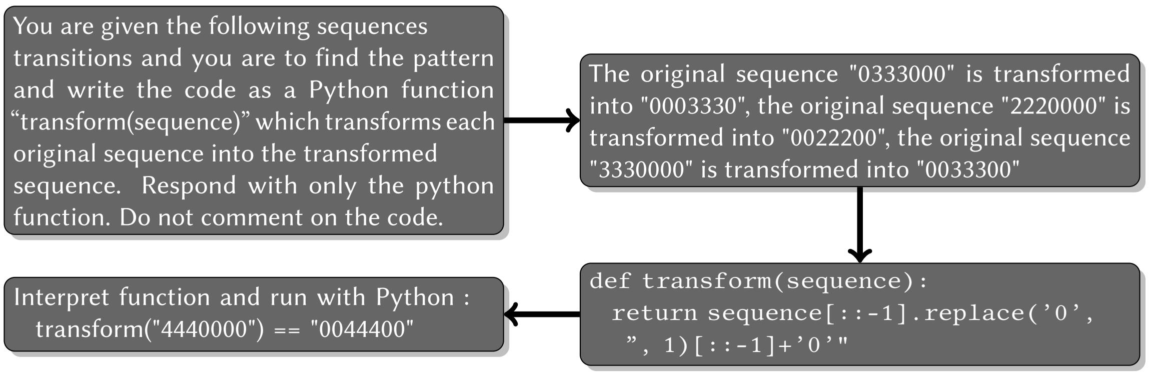
Firstly we encode the sequences as strings of integers, 0 representing black and other digits for other colors. We prompt an LLM to produce a "transform (sequence)" Python function which returns the transformed sequence. The Python function is generated given the few transformation examples provided in the prompt.

2.3. Prompt engineering

We experimented on a trial-and-error basis by interacting with the GPT 3.5/4 models directly in order to engineer a prompt which produces a usable function without producing too much unnecessary text. In fact, the GPT models tend to add many comments in their code even for very simple tasks which increases the produced tokens and therefore the cost. The OpenAI API for GPT models requires a "system" input, in which we describe the purpose of the query and what we expect as an output and a "user" entry in which we give the transformation examples. The API returns a response as JSON which includes the answer to our query, the number of input and output tokens and multiple choices depending on how many we asked for. We built three different prompting techniques, namely standard prompting, CoT and direct feedback which we describe next and provide comparative results.

2.3.1. Standard prompting

Our first method is standard prompting, namely prompting the LLM solely for a function solving the task. Figure 2 is an example of the generated prompt for task 1dMove2p7, the response from GPT-4 and the test we perform. No help is given to the LLM, clue or hint, solely the sequences of the transformations examples.



You are given the following sequences transitions and you are to find the pattern and write the code as a Python function "transform(sequence)" which transforms each original sequence into the transformed sequence. Respond with only the python function. Do not comment on the code.

The original sequence "0333000" is transformed into "0003330", the original sequence "2220000" is transformed into "0022200", the original sequence "3330000" is transformed into "0033300"

```
def transform(sequence):
  return sequence[::-1].replace(’0’,
   ”, 1)[::-1]+’0’"
```

Interpret function and run with Python :
  transform("4440000") == "0044400"

Figure 2: Standard prompting of the task 1dMove2p7, the answer from GPT-4 and the test we run.

Since this is the less costly of our methods, for each of the tasks we prompt both GPT-3.5-turbo and GPT-4 for 10 choices using standard prompting.

2.3.2. Chain of Thought

The CoT concept is to prompt the LLM to produce reasoning steps or decompose a task or question before giving an answer, since those produced tokens become the context, it is thought that it increases the accuracy of the answer given and has been used in various contexts.

There are multiple ways to perform CoT. One could give an example of the expected reasoning process, prompting for a "step-by-step" list, for reasoning in a controlled or free-form manner. As shown in Figure 3, we chose to add to our standard prompting method the prompt: "Provide your reasoning step by step prior to giving the function.".

The investigated LLMs do produce a description of the transformation and of the task to complete before writing the Python function. We will see in the results section how this affects the success rate as well as the cost of running the experiment for all 900 tasks.

2.3.3. Direct feedback

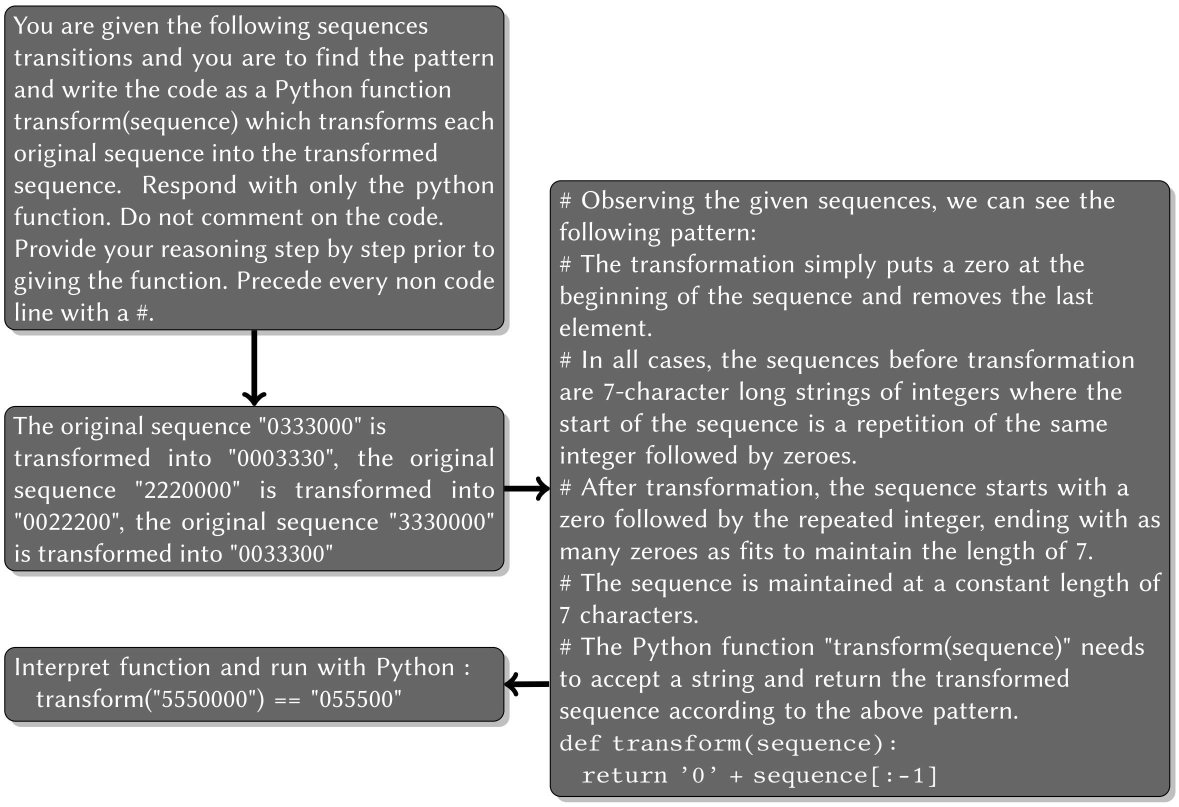
Seeing that many of the functions produced either do not run or fail at completing the tasks, we designed a different method called direct feedback in which we iteratively query GPT models, letting it know of previous functions on the same task which did not pass the test. Figure 4 shows an example of the prompt we generate on a second iteration once a function has failed. When a function succeeds at the task we stop iterating. We also stop if we reach the maximum number of iterations, in our experiment is 5.

The functions which failed the tasks are given in the prompt following the sentence: "Knowing that these functions failed:".

Direct feedback being iterative is much more costly than standard prompting, especially when the size of the input increases with the number of functions that failed previously (the number of iterations).

2.4. Testing generated code

To test the functions responded by the GPT models we first process the response, removing any additional lines that would not be coded and removing comments. The code is then interpreted and if it

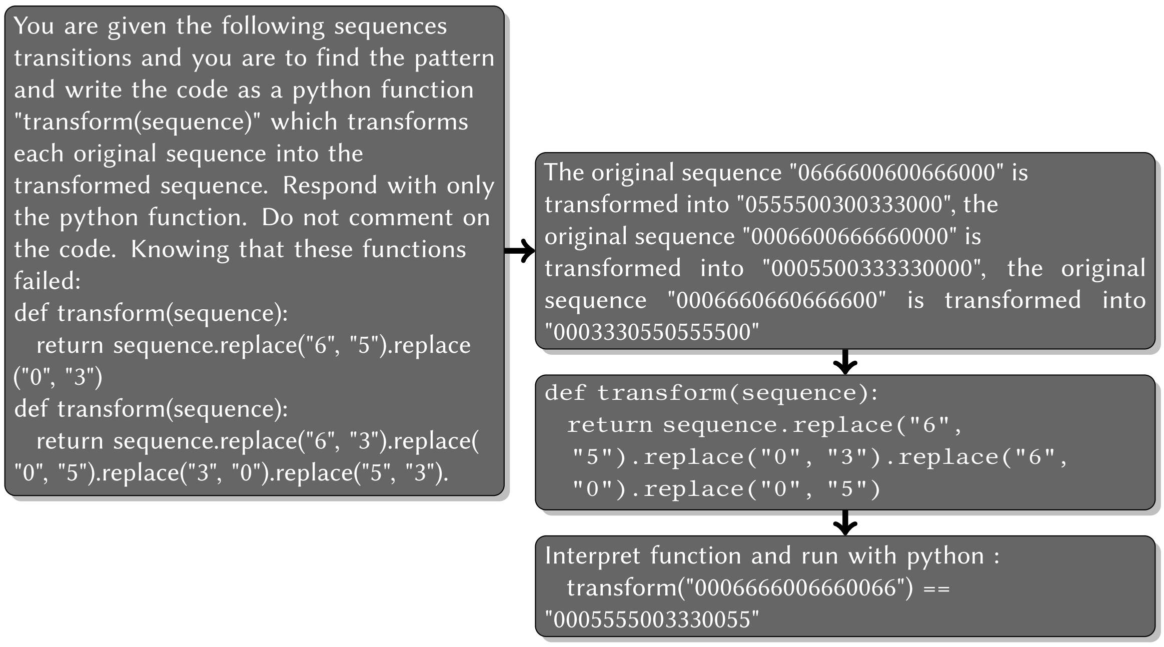

You are given the following sequences transitions and you are to find the pattern and write the code as a Python function transform(sequence) which transforms each original sequence into the transformed sequence. Respond with only the python function. Do not comment on the code. Provide your reasoning step by step prior to giving the function. Precede every non code line with a #.

The original sequence "0333000" is transformed into "0003330", the original sequence "2220000" is transformed into "0022200", the original sequence "3330000" is transformed into "0033300"

```
# Observing the given sequences, we can see the following pattern:
# The transformation simply puts a zero at the beginning of the sequence and removes the last element.
# In all cases, the sequences before transformation are 7-character long strings of integers where the start of the sequence is a repetition of the same integer followed by zeroes.
# After transformation, the sequence starts with a zero followed by the repeated integer, ending with as many zeroes as fits to maintain the length of 7.
# The sequence is maintained at a constant length of 7 characters.
# The Python function "transform(sequence)" needs to accept a string and return the transformed sequence according to the above pattern.
def transform(sequence):
  return '0' + sequence[:-1]
```

Interpret function and run with Python :
  transform("5550000") == "055500"

Figure 3: Generated CoT prompt for the task 1dMove1p44, the response from GPT-4 and the test we run.


You are given the following sequences transitions and you are to find the pattern and write the code as a python function "transform(sequence)" which transforms each original sequence into the transformed sequence. Respond with only the python function. Do not comment on the code. Knowing that these functions failed:
```
def transform(sequence):
  return sequence.replace("6", "5").replace("0", "3")
def transform(sequence):
  return sequence.replace("6", "3").replace("0", "5").replace("3", "0").replace("5", "3").
```

The original sequence "0666600600666000" is transformed into "0555500300333000", the original sequence "0006600666660000" is transformed into "0005500333330000", the original sequence "0006660660666600" is transformed into "0003330550555500"

```
def transform(sequence):
  return sequence.replace("6",
  "5").replace("0", "3").replace("6",
  "0").replace("0", "5")
```

Interpret function and run with python :
  transform("0006666006660066") == "0005555003330055"

Figure 4: Direct feedback iteration for the task 1dMove2p7, the answer from GPT-4 and the test we run.

can be interpreted will be accessible as a function. We then run this function to test if the transformation is correct by comparing the returned sequence with the output sequence from the dataset. We have guardrails when running the code, for instance, some functions run into infinite loops which we kill after 1 second. We store all responses from GPT-3.5/4, results, and calculations on Zenodo[1] for reference.

[1]Our implementation is available at https://zenodo.org/records/13735926. The repository also contains the 1D-ARC dataset from https://github.com/khalil-research/1D-ARC [19].

2.5. OpenAI API parameters

When querying the chat completion API we can specify parameters which affect the response generation. We adjust the number of choices to generate, when an LLM generates a completion, it selects the most likely next token according to a probability distribution on all tokens, OpenAI provides multiple choices resulting from choosing within those distributions. We set the temperature to 1 for all experiments. We did not specify a maximum number of tokens.

2.6. Evaluation metrics

To evaluate and compare our different prompting strategies, we compute metrics on the data resulting from testing, i.e. the number of successful tasks per task category and per number of choices.

$$\mu = \sum_{t \in \mathtt{tasks}} \frac{\mathtt{success}(t)}{|\mathtt{tasks}|} \tag{1}$$

We define the mean success rate μ as the sum of successful tasks divided by the total number of tasks. The cost c per method and number of choices is defined in USD by OpenAI as $\mathtt{input\_tokens} * 0.0000005 + \mathtt{output\_tokens} * 0.0000015$ with the GPT-3.5-turbo model and $\mathtt{input\_tokens} * 0.00003 + \mathtt{output\_tokens} * 0.00006$ with GPT-4.

We define the benefit-cost ratio as the mean success rate divided by the cost. We calculate it for each prompting strategy and number of choices/iterations.

$$\mathtt{bcr} = \frac{\mu}{c} \tag{2}$$

The following section analyses the results of our experiments.

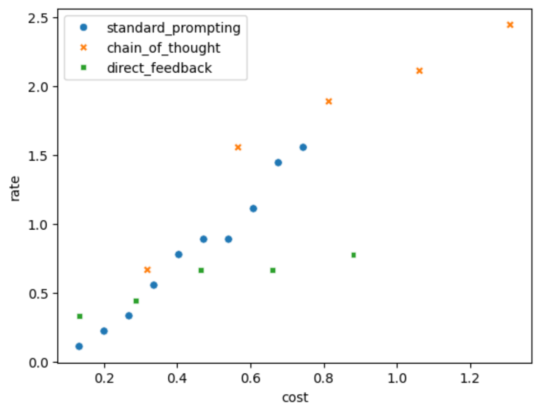
3. Results

We have queried the OpenAI API with both the GPT-3.5-turbo and GPT-4 models. The tables summarising the results are given in the appendix for clarity. With either model CoT performs better for the same number of choices but standard prompting reaches a similar success rate whilst remaining less costly. Even with GPT-4 and CoT, the LLM did not find a solution for any task of several task categories, namely Pattern Copy, Pattern Copy Multicolor and Mirror which do not seem as such complex tasks for a human, the most successful tasks are not necessarily the simplest, since the LLM is more successful at the Move by 2 pixels tasks than by 1 pixel, we think that there is a large amount of chance in solving the task hence why there is no clear rate per complexity of task correlation emerging from the results which would reveal some inductive reasoning from the LLM.

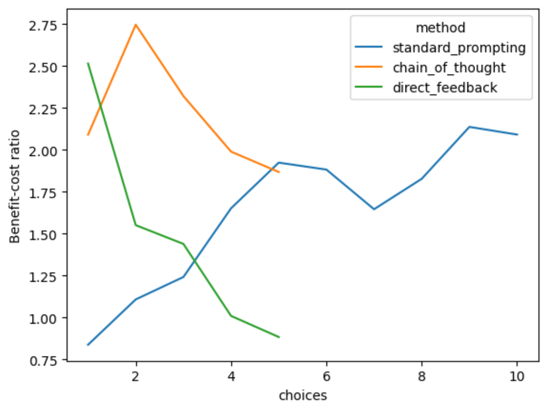
3.1. GPT-3.5-Turbo results

Tables 1, 2, 3 give the number of successful tasks out of 50 tasks per category. Tables 4, 5, 6 give the number of tokens used for input and output per amount of choices as well as the calculation of the mean success rate, the cost and benefit-cost ratio. The best μ is with 5 choices. The CoT strategy has a 2.44% success rate, which is really low, but slightly better than 10 choices with the standard prompting approach, which has a 1.56% success rate. The best *bcr* is with the CoT strategy and 2 choices, the *bcr* drops with more than 2 choices queried.

Figure 5a and 5b represent visually the results. The success rates and costs discussed in figure 5a show tradeoffs for all prompting approaches. However, the slope of the tradeoff for the direct feedback approach is less steep than the rest of the approaches. This means that even if the cost increases significantly, the increase in success rate is not that significant for the direct feedback approach.

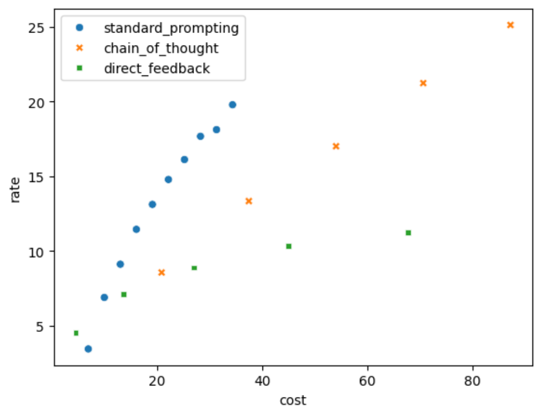


(a) Tradeoff analysis between success rate and cost.

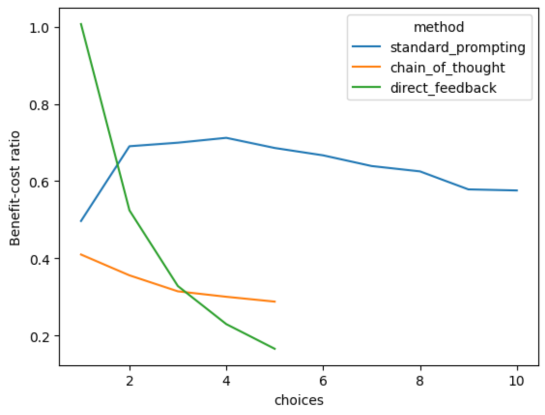


(b) Benefit-cost ratio per number of choices.

Figure 5: Analysis of ChatGPT-3.5-turbo results.



(a) Tradeoff analysis between success rate and cost.



(b) Benefit-cost ratio per number of choices.

Figure 6: Analysis of GPT-4 results.

3.2. GPT-4

Tables 7, 8, 9 give the number of successful tasks per category, out of 50 tasks per category. Tables 10, 11, 12 give the number of tokens used for input and output per amount of choices as well as the calculation of the mean success rate, the cost and benefit-cost ratio.

Figure 6a and 6b represent the results visually. Contrarily with the GPT-3.5 model, the best *bcr* is with 1 iteration of direct feedback whereas the *bcr* of standard prompting maximises at 4 choices. The CoT success rate is lower and decreases with the number of choices. The best success rate obtained is with 5 choices and the CoT strategy with 25.11% whereas standard prompting with 10 choices reaches 19.77%. However, the standard prompting is generally much less costly.

Figure 6a shows that the tradeoff slope for standard prompting is the steepest, followed by the CoT and direct feedback prompting approaches. As in the case of the GPT-3.5 model, in the GPT-4 model as well, a significant increase in the cost does not guarantee a significant increase in the success rates with the direct feedback approach.

4. Discussion

We observe generally low results with the LLMs, GPT-4 and GPT-3.5-turbo. GPT-4 obtained the best successful rate of around 25%, although greatly better (about 10 times) than the results compared to GPT-3.5-turbo. This suggests that the greater parameter size enhanced their capability with inductive reasoning with the prompts. While the results are still low, the performance of GPT-4 is encouraging. This shows the potential of prompts (i.e., forward propagation) of very large neural networks of billions of artificial neurons to approximate inductive reasoning. Further ways of learning with the prompts, e.g., instruction tuning and reinforcement learning, may help improve the performance.

The generally low results are also likely due to our numeric representation (e.g. “000333000”) of the 1D-ARC task, given that the tokenisation process may split the numeric string (e.g., by splitting it into “000”, “333”, “000”, separately), and thus may distort the meaning of the original string. Future studies can improve the numeric representation when prompting the model, or using other open LLMs (e.g., the Llama series [20]) which have a distinct processing of numeric strings.

We have run the standard prompting experiment with a Python list of integer embedding of the sequences, such as in [0,0,0,3,3,3,0,0,0] instead of “000333000” to compare and in fact there is an improvement, the success rate at 10 choices is 5% compared to 1.55% with GPT-3.5, 45 tasks out of 900 for a cost of $1.11 as opposed to $0.74. This result remains low and therefore shows the limited inductive reasoning capability of LLMs.

It might be possible to increase the performance of LLMs at coding for inductive reasoning by developing prompting techniques and we have shown that zero-shot CoT does improve the success rate. Nevertheless, the transformer architecture of LLMs and the autoregressive approach does not enable effective inductive reasoning and research in this direction is necessary.

5. Conclusion and Future Work

In this work, we have explored the capability of off-the-shelf LLMs, especially GPT-3.5 and GPT-4 for inductive reasoning using the 1D-ARC corpus. Results show great room for improvement for future systems that employ LLM. Program synthesis has been used as an intermediary task to solve the problem. While we mainly explored pure inductive reasoning with LLMs, the area of program synthesis has been explored greatly with both inductive and deductive methods. These methods can be explored in the future to generate programmable hypotheses to solve the tasks from the Abstract Reasoning Corpus.

Finally, the low performance of a pure LLM-based approach in this work may suggest the need for future studies to combine inductive and deductive methods with large neural networks like LLMs. For example, Retrieval Augmented Generative (RAG) [21] together with symbolic representation in deductive reasoning (e.g., graph-based RAG) and other LLM adaptation methods like fine-tuning with reinforcement learning may provide new avenues to combine neural-based inductive and deductive methods.

References

[1] T. Zhang, F. Ladhak, E. Durmus, P. Liang, K. McKeown, T. B. Hashimoto, Benchmarking large language models for news summarization, Transactions of the Association for Computational Linguistics 12 (2024) 39–57.

[2] M. Bommarito II, D. M. Katz, Gpt takes the bar exam, arXiv preprint arXiv:2212.14402 (2022).

[3] OpenAI, Gpt-4 technical report, 2023. arXiv:2303.08774.

[4] M. R. Morris, J. Sohl-dickstein, N. Fiedel, T. Warkentin, A. Dafoe, A. Faust, C. Farabet, S. Legg, Levels of agi: Operationalizing progress on the path to agi, arXiv preprint arXiv:2311.02462 (2023).

[5] O. Häggström, Artificial general intelligence and the common sense argument, in: Conference on Philosophy and Theory of Artificial Intelligence, Springer, 2021, pp. 155–160.

[6] B. Wang, X. Yue, H. Sun, Can chatgpt defend its belief in truth? evaluating llm reasoning via debate, in: Findings of the Association for Computational Linguistics: EMNLP 2023, 2023, pp. 11865–11881.

[7] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, et al., Chain-of-thought prompting elicits reasoning in large language models, Advances in Neural Information Processing Systems 35 (2022) 24824–24837.

[8] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, K. Narasimhan, Tree of thoughts: Deliberate problem solving with large language models, arXiv preprint arXiv:2305.10601 (2023).
[9] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al., Language models are few-shot learners, Advances in neural information processing systems 33 (2020) 1877–1901.
[10] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, Y. Iwasawa, Large language models are zero-shot reasoners, Advances in neural information processing systems 35 (2022) 22199–22213.
[11] S. J. Han, K. J. Ransom, A. Perfors, C. Kemp, Inductive reasoning in humans and large language models, Cognitive Systems Research 83 (2024) 101155.
[12] S. Imani, L. Du, H. Shrivastava, Mathprompter: Mathematical reasoning using large language models, arXiv preprint arXiv:2303.05398 (2023).
[13] K. Wang, H. Ren, A. Zhou, Z. Lu, S. Luo, W. Shi, R. Zhang, L. Song, M. Zhan, H. Li, Mathcoder: Seamless code integration in llms for enhanced mathematical reasoning, arXiv preprint arXiv:2310.03731 (2023).
[14] Z. Zhao, W. S. Lee, D. Hsu, Large language models as commonsense knowledge for large-scale task planning, Advances in Neural Information Processing Systems 36 (2024).
[15] S. Wang, Z. Wei, Y. Choi, X. Ren, Can llms reason with rules? logic scaffolding for stress-testing and improving llms, arXiv preprint arXiv:2402.11442 (2024).
[16] R. Wang, E. Zelikman, G. Poesia, Y. Pu, N. Haber, N. D. Goodman, Hypothesis search: Inductive reasoning with language models, 2023. arXiv:2309.05660.
[17] A. Kalyan, A. Mohta, O. Polozov, D. Batra, P. Jain, S. Gulwani, Neural-guided deductive search for real-time program synthesis from examples, in: International Conference on Learning Representations, 2018, pp. 1–15.
[18] J. Liu, C. S. Xia, Y. Wang, L. ZHANG, Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation, in: A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, S. Levine (Eds.), Advances in Neural Information Processing Systems, volume 36, Curran Associates, Inc., 2023, pp. 21558–21572.
[19] Y. Xu, W. Li, P. Vaezipoor, S. Sanner, E. B. Khalil, LLMs and the abstraction and reasoning corpus: Successes, failures, and the importance of object-based representations, 2023. arXiv:2305.18354.
[20] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al., Llama: Open and efficient foundation language models, arXiv preprint arXiv:2302.13971 (2023).
[21] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, D. Kiela, Retrieval-augmented generation for knowledge-intensive nlp tasks, in: H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems, volume 33, Curran Associates, Inc., 2020, pp. 9459–9474.

Appendix

Table 1
Standard prompting number of successful functions per task category out of 50 by the number of choices with GPT-3.5-turbo. The categories of Padded Fill, Move 2 Towards, Flip, Mirror, Denoise, Denoise Multicolor, Pattern Copy, Pattern Copy Multicolor, Recolor by Odd Even, Recolor by Size, Recolor by Size Comparison, and Scaling were omitted as none of the tasks were successful.

Choice	1	2	3	4	5	6	7	8	9	10
Task category										
Move 1	1	1	1	2	2	2	2	2	2	3
Move 2	0	0	0	0	0	0	0	1	2	2
Move 3	0	1	2	2	3	3	3	4	4	4
Move Dynamic	0	0	0	0	0	1	1	1	2	2
Fill	0	0	0	1	2	2	2	2	2	2
Hollow	0	0	0	0	0	0	0	0	1	1
Mean success rate (%)	**0.11**	**0.22**	**0.33**	**0.56**	**0.78**	**0.89**	**0.89**	**1.11**	**1.44**	**1.56**
Standard deviation	**0.47**	**0.64**	**1.02**	**1.33**	**1.83**	**1.84**	**1.84**	**2.19**	**2.35**	**2.52**

Table 2
Direct feedback number of successful functions per task category out of 50 by the number of choices with GPT-3.5-turbo. The categories of Move 2, Move 2 Towards, Move Dynamic, Fill, Padded Fill, Flip, Mirror, Denoise, Denoise Multicolor, Pattern Copy, Pattern Copy Multicolor, Recolor by Odd Even, Recolor by Size, Recolor by Size Comparison, and Scaling were omitted since none of those tasks were successful.

Choice	1	2	3	4	5
Task category					
Move 1	0	1	2	2	2
Move 3	2	2	2	2	3
Hollow	1	1	2	2	2
Mean success rate (%)	**0.33**	**0.44**	**0.67**	**0.67**	**0.78**
Standard deviation	**1.029**	**1.097**	**1.534**	**1.534**	**1.833**

Table 3
CoT number of successful functions per task category out of 50 by the number of choices with GPT-3.5-turbo. The categories of Pattern Copy, Mirror, Pattern Copy Multicolor, Recolor by Odd Even, Recolor by Size, Recolor by Size Comparison, and Scaling were omitted since none of those tasks were successful.

Choice	1	2	3	4	5
Task category					
Move 1	3	5	6	6	6
Move 2	0	1	1	1	1
Move 3	0	1	1	1	1
Move Dynamic	1	1	1	1	1
Move 2 Towards	0	0	0	0	1
Fill	1	2	2	2	2
Padded Fill	0	0	0	0	0
Hollow	1	2	3	5	7
Flip	0	1	1	1	1
Denoise	0	1	1	1	1
Denoise Multicolor	0	0	1	1	1
Mean success rate (%)	**0.67**	**1.56**	**1.89**	**2.11**	**2.44**
Standard deviation	**1.534**	**2.526**	**3.027**	**3.462**	**4.033**

Table 4
Standard prompting used tokens, success rate and costs per number of choices with GPT-3.5-turbo

Number of choices	1	2	3	4	5
Input tokens	**129607**	**129607**	**129607**	**129607**	**129607**
Output tokens	**45289**	**90578**	**135867**	**181156**	**226446**
Mean success rate (%)	**0.111**	**0.222**	**0.333**	**0.556**	**0.778**
Cost in $	**0.133**	**0.201**	**0.269**	**0.337**	**0.404**
Benefit-cost ratio	**0.837**	**1.107**	**1.241**	**1.651**	**1.923**
Number of choices	6	7	8	9	10
Input tokens	**129607**	**129607**	**129607**	**129607**	**129607**
Output tokens	**271735**	**317024**	**362313**	**407602**	**452891**
Mean success rate (%)	**0.889**	**0.889**	**1.111**	**1.444**	**1.556**
Cost in $	**0.472**	**0.54**	**0.608**	**0.676**	**0.744**
Benefit-cost ratio	**1.882**	**1.645**	**1.827**	**2.136**	**2.09**

Table 5
CoT used tokens, success rate and costs per number of choices with GPT-3.5-turbo

Number of choices	1	2	3	4	5
Input tokens	**143122**	**143122**	**143122**	**143122**	**143122**
Output tokens	**165003**	**330007**	**495010**	**660014**	**825017**
Mean success rate (%)	**0.667**	**1.556**	**1.889**	**2.111**	**2.444**
Cost in $	**0.319**	**0.567**	**0.814**	**1.062**	**1.309**
Benefit-cost ratio	**2.089**	**2.746**	**2.32**	**1.989**	**1.867**

Table 6
Direct feedback used tokens, success rate and costs per number of choices with GPT-3.5-turbo

Number of choices	1	2	3	4	5
Input tokens	**129964**	**308737**	**528124**	**789198**	**1092075**
Output tokens	**45080**	**88221**	**132960**	**177407**	**223007**
Mean success rate (%)	**0.333**	**0.444**	**0.667**	**0.667**	**0.778**
Cost in $	**0.133**	**0.287**	**0.464**	**0.661**	**0.881**
Benefit-cost ratio	**2.514**	**1.55**	**1.438**	**1.009**	**0.883**

Table 7
Standard prompting number of successful functions per task category out of 50 by the number of choices with GPT-4. The categories of Pattern Copy, Pattern Copy Multicolor, Mirror, Recolor by Odd Even, Recolor by Size, and Recolor by Size Comparison were omitted since none of those tasks were successful.

Choice	1	2	3	4	5	6	7	8	9	10
Task category										
Move 1	7	11	13	14	17	19	20	21	21	25
Move 2	2	4	4	5	5	5	5	7	8	9
Move 3	2	8	12	13	13	14	16	18	18	18
Move Dynamic	2	2	2	3	4	4	4	6	6	6
Move 2 Towards	0	0	0	0	0	1	1	2	2	3
Fill	5	13	19	22	22	24	26	27	28	30
Padded Fill	0	0	0	0	0	1	1	1	1	1
Hollow	6	10	14	18	22	26	29	30	30	32
Flip	5	7	9	13	19	21	22	24	24	27
Denoise	0	4	4	5	5	5	6	6	6	7
Denoise Multicolor	1	1	1	4	5	6	8	10	12	13
Scaling	1	2	4	6	6	7	7	7	7	7
Mean success rate %	**3.44**	**6.89**	**9.11**	**11.44**	**13.11**	**14.78**	**16.11**	**17.67**	**18.11**	**19.78**
Standard deviation	**4.74**	**8.81**	**12.17**	**14.25**	**16.38**	**18.28**	**19.88**	**20.81**	**21.04**	**22.93**

Table 8
CoT number of successful functions per task category out of 50 by the number of choices with GPT-4. The categories of Pattern Copy, Pattern Copy Multicolor, and Mirror were omitted since none of those tasks were successful.

Choice	1	2	3	4	5
Task category					
Move 1	7	10	12	14	18
Move 2	5	15	17	22	27
Move 3	6	8	10	15	19
Move Dynamic	2	2	2	4	6
Move 2 Towards	1	2	3	3	4
Fill	5	11	14	17	23
Padded Fill	1	1	1	1	1
Hollow	16	25	28	37	39
Flip	9	11	20	21	24
Denoise	5	8	12	13	16
Denoise Multicolor	15	21	24	31	34
Recolor by Size	1	2	2	2	2
Recolor by Size Comparison	0	0	1	1	1
Scaling	4	4	7	10	12
Mean success rate (%)	**8.56**	**13.33**	**17.0**	**21.22**	**25.11**
Standard deviation	**9.889**	**15.262**	**18.153**	**22.949**	**25.743**

Table 9
Direct feedback number of successful functions per task category out of 50 by the number of choices with GPT-4. The categories of Mirror, Pattern Copy, Pattern Copy Multicolor, Padded Fill, Recolor by Odd Even, Recolor by Size, and Recolor by Size Comparison were omitted since none of those tasks were successful.

Choice	1	2	3	4	5
Task category					
Move 1	5	10	13	14	17
Move 2	1	2	4	4	4
Move 3	4	5	5	6	6
Move Dynamic	1	1	1	3	3
Move 2 Towards	1	1	2	3	3
Fill	9	13	15	15	16
Hollow	10	15	20	23	23
Flip	5	8	9	11	11
Denoise	1	1	2	4	5
Denoise Multicolor	3	4	5	6	6
Scaling	1	4	4	4	7
Mean success rate (%)	**4.56**	**7.11**	**8.89**	**10.33**	**11.22**
Standard deviation	**6.28**	**9.634**	**11.985**	**13.092**	**13.791**

Table 10
Standard prompting used tokens, success rate and costs per number of choices with GPT-4

Number of choices	1	2	3	4	5
Input tokens	**129607**	**129607**	**129607**	**129607**	**129607**
Output tokens	**50767**	**101533**	**152300**	**203067**	**253834**
Mean success rate (%)	**3.444**	**6.889**	**9.111**	**11.444**	**13.111**
Cost in $	**6.934**	**9.98**	**13.026**	**16.072**	**19.118**
Benefit-cost ratio	**0.497**	**0.69**	**0.699**	**0.712**	**0.686**
Number of choices	6	7	8	9	10
Input tokens	**129607**	**129607**	**129607**	**129607**	**129607**
Output tokens	**304600**	**355367**	**406134**	**456900**	**507667**
Mean success rate (%)	**14.778**	**16.111**	**17.667**	**18.111**	**19.778**
Cost in $	**22.164**	**25.21**	**28.256**	**31.302**	**34.348**
Benefit-cost ratio	**0.667**	**0.639**	**0.625**	**0.579**	**0.576**

Table 11
CoT used tokens, success rate and costs per number of choices with GPT-4

Number of choices	1	2	3	4	5
Input tokens	**143122**	**143122**	**143122**	**143122**	**143122**
Output tokens	**276249**	**552498**	**828748**	**1104997**	**1381246**
Mean success rate (%)	**8.556**	**13.333**	**17.0**	**21.222**	**25.111**
Cost in $	**20.869**	**37.444**	**54.019**	**70.593**	**87.168**
Benefit-cost ratio	**0.41**	**0.356**	**0.315**	**0.301**	**0.288**

Table 12
Direct feedback used tokens, success rate and costs per number of choices with GPT-4

Number of choices	1	2	3	4	5
Input tokens	**129607**	**309496**	**541517**	**828076**	**1168956**
Output tokens	**10624**	**71260**	**180380**	**335254**	**542357**
Mean success rate (%)	**4.556**	**7.111**	**8.889**	**10.333**	**11.222**
Cost in $	**4.526**	**13.56**	**27.068**	**44.957**	**67.61**
Benefit-cost ratio	**1.007**	**0.524**	**0.328**	**0.23**	**0.166**