LLMs as a System of Multiple Expert Agents: An Approach to Solve the Abstraction and Reasoning Corpus (ARC) Challenge

John Chong Min Tan*, Mehul Motani†
*†Department of Electrical and Computer Engineering, †Institute of Data Science, †N.1 Institute for Health,
†Institute for Digital Medicine (WisDM), National University of Singapore
Email: *johntancm@u.nus.edu, †motani@nus.edu.sg

***Abstract*—We attempt to solve the Abstraction and Reasoning Corpus (ARC) Challenge using Large Language Models (LLMs) as a system of multiple expert agents. Using the flexibility of LLMs to be prompted to do various novel tasks using zero-shot, few-shot, context-grounded prompting, we explore the feasibility of using LLMs to solve the ARC Challenge. We firstly convert the input image into multiple suitable text-based abstraction spaces. We then utilize the associative power of LLMs to derive the input-output relationship and map this to actions in the form of a working program, similar to Voyager / Ghost in the MineCraft. In addition, we use iterative environmental feedback in order to guide LLMs to solve the task. Our proposed approach achieves 50 solves out of 111 training set problems (45%) with just three abstraction spaces - grid, object and pixel - and we believe that with more abstraction spaces and learnable actions, our approach will be able to solve more.**

***Index Terms*—Abstraction and Reasoning Corpus, ARC Challenge, Multiple Agents, Multiple Experts, Mass Sampling and Filtering, Abstraction Spaces, Primitive Functions, Iterative Feedback**

I. Introduction

The Abstraction and Reasoning Corpus (ARC) Challenge is a key milestone in the march towards artificial general intelligence (AGI) as it requires forming concepts and abstractions [1]. Fig. 1 illustrates a sample ARC task. One of the key difficulties of the ARC challenge is that it requires doing something counter to mainstream deep learning – learning from very few samples. Deep learning typically uses tens of thousands of samples to do well. Humans, in comparison, can learn how to identify different animals by just one or two different observations. For instance, a child can identify a giraffe in real life for the first time, even though the only other time they may have been exposed to a giraffe was through a cartoon flash card. Such capabilities are not well endowed in modern AI systems, and that means that such AI systems will need to be trained extensively before deploying in the real world. After deploying them in the real world, they will also be limited in their ability to adapt and learn as the environment changes.

In contrast, traditional symbol-based systems (e.g., GOFAI [2]) can "learn" quite fast, as any new situation can be interpreted without any learning phase, provided that there are existing symbols which can represent it. However, the history of GOFAI has shown that it is difficult to engineer

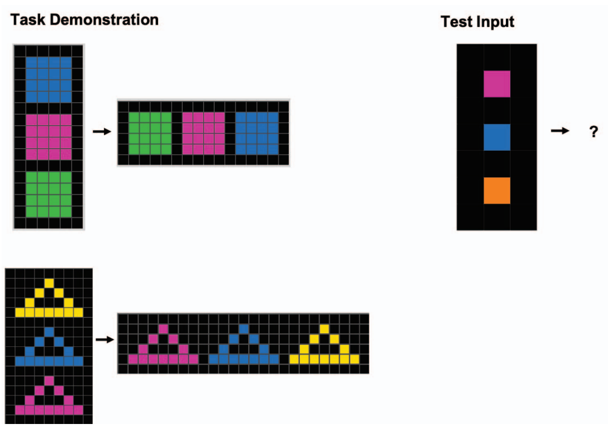

Fig. 1. A sample ARC task. The challenge is to infer the abstract rule(s) governing the demonstration transformations and apply it to the test input. Example from: https://aiguide.substack.com/p/why-the-abstraction-and-reasoning

these symbols, and at many times, even humans face difficulty to come up with symbols as they may not be able to express it in words.

As can be seen, there are shortcomings with the above two approaches, and a new kind of approach will be needed in order to learn fast and generalise to new situations, in order to even have a chance at solving the ARC Challenge. In this paper, we address this challenge by proposing to use Large Language Models (LLMs) as a system grounded in functional action spaces to tackle the ARC challenge. This can be said to be an intermediate ground between both deep learning and GOFAI approaches - the functional action spaces are more flexible than symbols in GOFAI; LLMs are a form of deep learning that are adaptable to new situations via prompting. Specifically the contributions of the paper are as follows:

- We showcase a novel method of using LLMs as a system of multiple expert agents (without any pre-training) to solve the ARC Challenge
- We highlight the importance of a combination of multiple abstraction spaces from which to associate the input space to the output space
- We demonstrate the feasibility of grounding in functional space for program synthesis by LLMs.

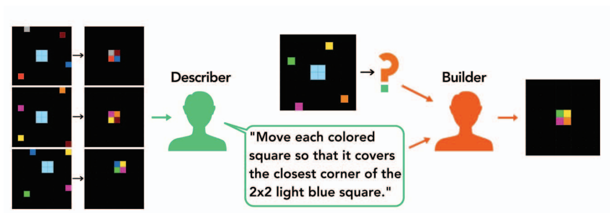
Describer
Builder
"Move each colored square so that it covers the closest corner of the 2x2 light blue square."

Fig. 2. 88% of ARC tasks can be solved by the Builder from just the description alone given by the Describer, without input-output examples. Can GPT-4 function as both the describer and the builder? Image reproduced from Fig. 4 of [3].

II. Related Work

ARC Challenge. The ARC challenge [1] comprises 400 public training tasks, 400 public evaluation tasks and 200 private test tasks. Each of these tasks has multiple "Task Demonstration" Input/Output grids, of which the task-taker must infer a common relation out of them. This common relation is then applied to the "Test Input", from which we get the "Test Output". The "Test Output" must match perfectly for it to be considered solved. The grids comprise grid sizes of 1x1 to 30x30, of which pixels can take on 10 different values.
Domain Specific Language (DSL) Approaches. The majority of ARC Challenge solutions are mainly DSL ones [4; 5; 6]. This is also the case for the first-place solution of the ARC Kaggle competition (https://www.kaggle.com/code/icecuber/arc-1st-place-solution).
LLM-Based approaches. One way to approach the ARC challenge will be to use text to describe the visual characteristics of objects [7]. Indeed, 88% of ARC tasks can be solved via language description alone without input-output examples as shown in Fig. 2 [3]. For certain problems, denoting pixels in terms of objects can significantly boost the solve rate from 13 to 23 out of 50 object-related ARC tasks [8]. Some work has also been done to do end-to-end input to program description generation with just LLMs alone to some success [9]. Other approaches have used Decision Transformers [10] to find a sequence of primitive actions from the input to output [11], however, as noted by the authors, huge amounts of data (10000 training data for 2000 testing data) are needed to train this method, it is unlikely it can generalise to unseen inputs. Recently, LLMs have been used to take the ASCII text view of the grid as input for next token prediction and have solved 85 out of 800 ARC tasks [12].
Code as Skills and Environmental Feedback. Voyager is an embodied lifelong learning agent powered by LLMs [13]. It features a skill library of functions to build up complex behaviour, and an iterative prompting mechanism with the environment to learn from environmental feedback. Ghost in the Minecraft [14] does something similar as well, though they constrain the action space to a list of functions. Similarly, we use code generation with primitive functions to approximate using a skill library, and use iterative prompting using ARC task output as feedback to learn from the environment.

Our Method. In line with the existing LLM approaches, we agree that we should use language as an alternate abstraction space in addition to the original pixel grid. Unlike existing approaches, we believe we should use more than one abstraction space. Hence, the LLM will be both the Builder and the Describer in Fig. 2, but the Builder can also reference input-output pairs. We also believe we should integrate LLMs with a kind of DSL approach, but can afford to have an even more expressive DSL because an LLM is able to do matching of functions via semantics much more effectively than traditional DSL approaches.

III. Broad Overview of Method

In this section, we provide an overview of our proposed approach and envisioned future approach and discuss several key ideas behind it. Generative Pre-trained Transformer 4 (GPT-4) is a multimodal LLM created by OpenAI and released in March 2023 [15]. For now, we exclusively use GPT-4 for our model, as we empirically observe that GPT-3.5 and other open source models (e.g. Llama 2, Vicuna) are not able to perform well enough for this method to work. Empirically, we also observe that using GPT-4V as an image-based abstraction space is inferior to rule-based abstractions formed by our approach, due to errors interpreting the grid images. The current overall method is shown in Fig. 3 and the future envisioned system is shown in Fig. 4. The overall approach is detailed below, with parts not implemented yet highlighted as future work. Note that we do not need the parts highlighted as future work for the system to work - it will just make it more efficient and learn better from experience.
Problem Type Classification (Future Work). ARC tasks test various concepts. If we can use past examples to ground the LLM, and let the LLM decide what problem category an ARC task belongs to, we can proceed with a specialised workflow to target solving that particular style of task. Presently, we simply run through all the various agent types and select the agent types which work. Implementing this classifier will not affect performance but will significantly help reduce the costs.
Useful Abstraction Spaces. While GPT-4 has proven to be a general purpose solver, being primarily a text-based model, GPT-4 lacks some of the innate human priors necessary to solve the ARC challenge. For example, GPT-4 is not able to identify objects accurately from text alone. Objects are defined as continuous sections of the grid with the same non-zero value. Hence, providing such an object view as an abstraction space using text greatly helps with the GPT-4's ability to form associations with the input-output pair and is better able to find a solution [8]. Moreover, we can provide more than one abstraction space to GPT-4, which can increase the chance that one or more abstraction spaces contain a simple mapping from input to output, thereby reducing the complexity of the problem. Do note that these abstraction spaces are unchangeable, and are fixed since the beginning of learning. Hence, these form fixed priors for stable learning.
Encoding Human Biases via Helper/Primitive Functions. An initial implementation of using GPT-4 to solve ARC was

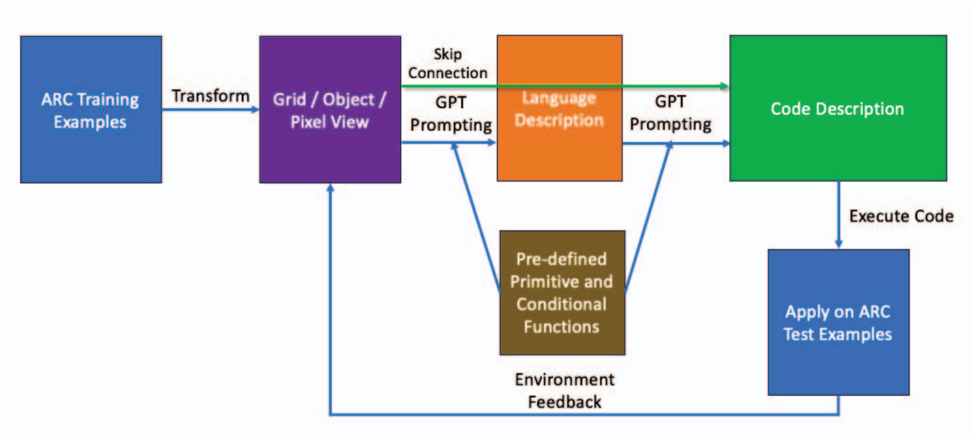

Fig. 3. Current Process Flowchart of LLMs as a System to solve the ARC Challenge.

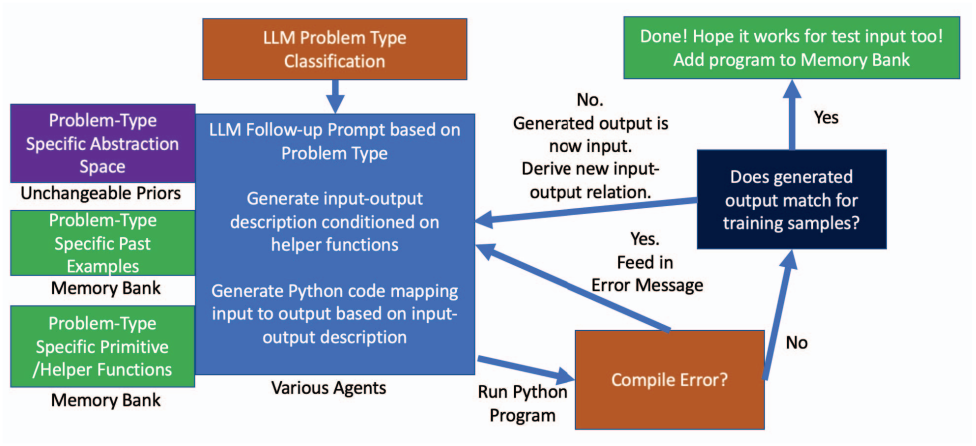
LLM Problem Type Classification
Done! Hope it works for test input too! Add program to Memory Bank
Problem-Type Specific Abstraction Space
Unchangeable Priors
Problem-Type Specific Past Examples
Memory Bank
Problem-Type Specific Primitive /Helper Functions
Memory Bank
LLM Follow-up Prompt based on Problem Type
Generate input-output description conditioned on helper functions
Generate Python code mapping input to output based on input-output description
Various Agents
No. Generated output is now input. Derive new input-output relation.
Yes
Does generated output match for training samples?
Yes. Feed in Error Message
No
Run Python Program
Compile Error?

Fig. 4. Future Envisioned Process Flowchart of LLMs as a System to solve the ARC Challenge.

done with just prompting the human biases and action spaces via text. This did not do so well due to lack of grounding using words alone. A key innovation in this work is to use primitive functions as action spaces, as a way to encode human priors. If we could use functions for grounding, and express the semantic meaning of the function in words, GPT-4 could use the function to provide the code needed for the solution. Hence, the problem now becomes finding out what are the primitive functions we need to encode in order for the LLM to solve any generic ARC problem.

Using Memory for Additional Context (Future Work). New problems might mix and match aspects of previous solutions, so having a memory bank to provide examples of similar solved problems in the past can help to ground the LLM to better generate the answer. This is currently not implemented due to constraints of context length. Once the context length for GPT-4 increases, we intend to let each agent have memory of relevant previously solved problems and their solutions, so that it can ground the agent's output. This is akin to Retrieval Augmented Generation (RAG) [16].

Utilising Feedback from Environment. Another key idea is that a learning system would need to utilise feedback from the environment, and so a recursive loop feeding in feedback from the environment (whether there is compile error, whether the code matches the intended output) can help a lot in getting the right answer. This is akin to what is done in Voyager and Ghost in the MineCraft [13; 14].

LLMs as a System. Humans do not operate with only one system. We have various systems to call for various tasks. Similarly, we can have multiple expert agents for each task (such as *Object View*, *Pixel View*, *Grid View*) and call on them to give their interpretation of the task, and select the most promising agent. This greatly helps narrow the search space for the solution. Then, we utilise the specialised functions this agent has and solve the problem. Interfacing this agent with environment feedback, the problem-type specific abstraction space, past examples and action spaces can greatly help filter and ground GPT-4 to generate a plausible solution. We believe that, with better grounding via expert agents, better abstraction space representations and better primitive function grounding, we will eventually be able to solve most of the ARC tasks using the proposed approach.

IV. Detailed Overview of Method

We now go into some details of our method.

A. *Different Abstraction Spaces*

We utilise various ways of encoding the abstraction spaces so that GPT-4 can better associate between the Input-Output pairs. It has been shown in Image-Joint Embedding Predictive Architecture (I-JEPA) [17] and Stable Diffusion [18] that prediction in the latent/abstraction space leads to better downstream tasks than predicting in the input space. However, instead of just one abstraction space, we believe that there are many possible abstraction spaces which are fixed, and it is up to the solver to choose which is the best for the task at hand. We believe by incorporating more useful views and refining current ones, we can solve more ARC tasks.

For our method, we use only three views - *Grid View*, *Object View*, *Pixel View* - and that has already achieved quite good results. In brief, *Grid View* provides the entire grid representation, except we change the pixel numbers to characters so that we do not bias GPT-4 to treat it as an arithmetic problem to perform arithmetic on the pixel values. This also has the added benefit of ensuring that GPT-4 has not seen the ARC tasks before as it is now of a different form. The *Object View* groups pixels that are contiguous together, so that they can be manipulated as a group. *Pixel View* gives the coordinates for each pixel, which can help with more fine-grained movement tasks or relational tasks between pixels.

B. *JSON-based output format*

LLMs are well known for being verbose and also relatively free-form in the output, making it hard for any automated program to use it. Here, we explicitly ask GPT-4 to output in a JSON format via prompting. This JSON format also facilities Chain-of-Thought (CoT) prompting [19], as it is done in a specific sequence to encourage broad to specific thinking.

C. *CoT Prompting*

CoT enables the output to be structured and the LLM will be able to condition the generation of the later output based on the earlier ones. This enables a more broad to specific style of prompting, helping the LLM to think and reflect on various areas, narrowing the search space, and ultimately may help to solve the problem.

Here, we do CoT prompting directly using JSON format. We ask GPT-4 to output:

1) "reflection": "reflect on the answer",
2) "pixel_changes": "describe the changes between the input and output pixels, focusing on movement or pattern changes",
3) "object_changes": "describe the changes between the input and output objects, focusing on movement, object number, size, shape, position, value, cell count",
4) "helper_functions": "list any relevant helper_functions for this task",
5) "overall_pattern": "describe the simplest input-output relationship for all input-output pairs",
6) "program_instructions": "Plan how to write the python function and what helper functions and conditions to use",
7) "python_program": "Python function named 'transform_grid' that takes in a 2D grid and generates a 2D grid. Output as a string in a single line with \n and \t."

D. *Helper/Primitive Functions*

For the functions, we basically zero-shot prompt by stating the function name plus the input parameters and the description of the function. We find that this format of zero-shot prompting works very well for most functions, especially if the name of the function is already indicative of what it does. This is very similar to the approach taken in Visual ChatGPT [20], as well as OpenAI Functions (https://openai.com/blog/function-calling-and-other-api-updates). As this method of prompting is not sufficient to imbue biases that are not inherent in text (i.e. rotation, flipping), we also provide one-shot examples of how to use the function.

E. *Conditional Functions:*

Rather than letting GPT-4 free-form generate its own code, we ask it to generate a conditional flow on the primitive functions. This greatly helps to reduce compilation errors. Such a conditional flow is needed, as some ARC tasks require using logic that only applies if a particular condition is met (e.g., turn the shape red if it has exactly 6 cells). Without this conditional flow, the program would need many more steps before it can solve the problem. An example of such a conditional flow is:
If $\{$*condition*$\}$: $\{$*Primitive Function*$\}$

V. Methodology

Select Problems by Context Length. We firstly filter the ARC training set problems to only those whose Grid View and Object View (mono-color, no diagonals) can fit into a context length of 3000 tokens. This is important because later when we incorporate environmental feedback, we will need additional token length, and by empirical observation, 3000 tokens is necessary to guarantee some buffer token amount so that the entire prompt can fit within 8000 tokens later. This is the current maximum context length for the GPT-4 web browser, as well as for the basic GPT-4 API. In the future, we envision that our approach can work for more ARC tasks when the context length for GPT-4 increases.
Mass Sampling and Filtering. Next, we use the OpenAI API for GPT-4 May 24 2023 version with a temperature of 0.7 to ensure a diverse range of outputs. We use the OpenAI API and the web browser interface for GPT-4 interchangeably. We

TABLE I
NUMBER OF TASKS SOLVED, NOT SOLVED AND PARTIALLY SOLVED (PROGRAM WORKS FOR TASK DEMONSTRATION BUT NOT FOR TEST INPUT/OUTPUT OUT OF 111 TRAINING SET TASKS).

Total Tasks	Tasks Solved	Tasks Not Solved	Tasks Partially Solved
111	50	58	3

TABLE II
TASKS NOT SOLVED BUT WITH CORRECT DESCRIPTION

Total Tasks Not Solved	Correct description
61	8

TABLE III
TASKS SOLVED WITH ITERATIVE FEEDBACK LOOP AFTER EITHER INCORRECT OUTPUT OR COMPILE ERROR

Total	Incorrect Output	Compile Error
50	6	1

employ a mass sampling and filtering process to generate code, much like in AlphaCode [21] (see Fig. 5). *Grid View* is always there unless there is context length limitation. We can choose between toggling *Object View* (10 types) and *Pixel View* for the agents (at least one must be active), which leads to a total of $10 \times 2 = 20$ agents. We utilise each expert agent three times each, with at most three feedback loop iterations, and filter the output codes which can solve the Task Demonstration to try it out on the Task Input. If there are multiple such codes, we randomly pick three to test it out. Any of these three solutions passing the Test Input will be counted as a solve, which is in line with the Kaggle competition and Lab 42's ARCathon.

VI. RESULTS

Overall. Overall, as shown in Table I, our method solves 50 out of 111 Training Set ARC tasks which could fit within the context length. This is about a 45% solve rate, which is quite remarkable as the current ARC world record solve rate is 30.5% (though this is on the hidden test set), according to https://lab42.global/arcathon/updates/.

Coding Issues. To see how many of the unsolved problems are due to coding issues, we check how many of them have the correct description as evaluated by a human, but not have the correct code. This turns out to be 8 out of 61, as shown in Table II. This means that if we could learn the primitive/helper functions better and have a wider range to choose from, we can improve solve rate. To solve the rest of the problems, we will have to incorporate better views - it is observed that GPT-4 cannot solve line continuation tasks, especially for diagonal lines, grid manipulation tasks, and symmetry tasks easily, and these could easily be incorporated as additional views.

Iterative Feedback. To see how much iterative environmental feedback helps, we look at number of tasks solved with the iterative environment feedback loop. This turns out to be 7 tasks out of 50, as shown in Table III. This is quite significant, and highlights the importance of environmental feedback.

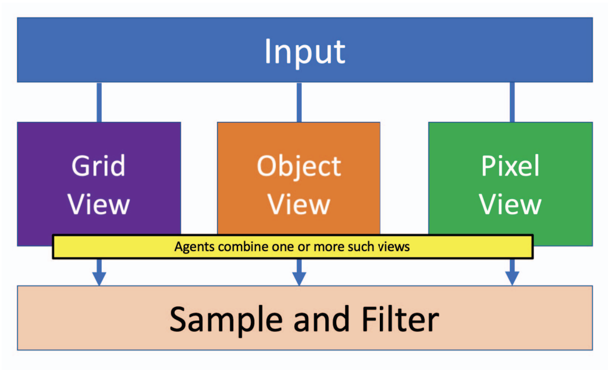

Fig. 5. Mass Sampling and Filtering process with various expert agents

VII. DISCUSSION

The results are promising, and GPT-4 agents with various combination of views can solve different types of problems well, as compared to just using the original *Grid View*. It was also sometimes observed that *Object View* had to go with *Pixel View* for a consolidation of information across both views in order to solve the task. This reinforces the view that there should not be just one, but multiple abstraction spaces which could be used in combination with each other.

Empirical observation has shown that GPT-4 with primitive function grounding can solve more tasks than without. It is a better way at encoding priors than with just text alone. Overall, GPT-4 is great at solving tasks which are made up of a combination of primitive functions.

It was observed that function names and descriptions are very important - GPT-4 tends to choose functions semantically similar to what it intends to do, and the changing of a function name to something irrelevant may cause it not to be used.

VIII. IMPROVEMENTS

GPT-4 agents cannot do tasks that have no relevant priors encoded in the primitive functions well, such as scaling of objects, symmetry, continuation of lines, overlay of grids with logical rules, grid manipulation like cropping, translating, changing of shape. Furthermore, it is weak when there is more than one relation, and this type of problems benefit from the iterative environment feedback loop. By setting the new input as the output that GPT-4's program outputs, it is in effect taking a step towards the solution and helps GPT-4 better associate the simpler input-output relationship.

GPT-4 has been observed to use primitive functions not meant for the view, for example, *Pixel View* Agent using the get_objects function. Hence, giving too much context might affect performance. This is similar to [8] when the performance declined after adding in relations between objects. We should limit each agent to only use relevant primitive functions.

IX. Conclusion and Future Work

Currently, we use all agents in a brute-force manner for a task. In order to reduce computation (and cost), we could perhaps have a classifier which takes in previous examples as input to learn how to classify a new problem into a category, so that the right agents can be used to solve it.

Currently, the primitive functions are hand-engineered based on observation of the first 50 tasks in the training set, and are also not a complete set. We will try to incorporate a way for GPT-4 to be prompted to create new primitive functions, and add those successful functions which could solve a new task to the list of primitive functions, much like Voyager [13]. One way is to add any transform_grid function that is successful as a new primitive function, as long as the description of the function is different from existing ones.

Overall, LLMs as a system of multiple expert agents with environmental feedback are a promising approach towards solving the ARC Challenge. Refer to https://arxiv.org/abs/2310.05146 for full details of the prompt, primitive functions, conditional functions, abstraction views, analysis of tasks solved and future proposed agent types. Our code can be found at https://github.com/tanchongmin/ARC-Challenge, and video explanation can be found at https://www.youtube.com/watch?v=plVRxP8hQHY.

Acknowledgements

This research is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG2-PhD-2021-01-003[T]) and by A*STAR, CISCO Systems (USA) Pte. Ltd and National University of Singapore under its Cisco-NUS Accelerated Digital Economy Corporate Laboratory (Award I21001E0002)

Many thanks to the various people who have provided encouragement to pursue the LLM route to solve ARC or have provided valuable insights - Pascal Kaufmann, Rolf Pfister, Michael Hodel, Simon Strandgaard, Douglas Miles, Richard Cottrill, Leonard Tan and many others.

References

[1] F. Chollet, “On the measure of intelligence,” *arXiv preprint arXiv:1911.01547*, 2019.

[2] M. A. Boden, “4 gofai,” *The Cambridge handbook of artificial intelligence*, p. 89, 2014.

[3] S. Acquaviva, Y. Pu, M. Kryven, C. Wong, G. E. Ecanow, M. Nye, T. Sechopoulos, M. H. Tessler, and J. B. Tenenbaum, “Communicating natural programs to humans and machines,” *arXiv preprint arXiv:2106.07824*, 2021.

[4] S. Alford, “A neurosymbolic approach to abstraction and reasoning,” Ph.D. dissertation, Massachusetts Institute of Technology, 2021.

[5] S. Ferré, “First steps of an approach to the arc challenge based on descriptive grid models and the minimum description length principle,” *arXiv preprint arXiv:2112.00848*, 2021.

[6] Y. Xu, E. B. Khalil, and S. Sanner, “Graphs, constraints, and search for the abstraction and reasoning corpus,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 4, 2023, pp. 4115–4122.

[7] G. Camposampiero, L. Houmard, B. Estermann, J. Mathys, and R. Wattenhofer, “Abstract visual reasoning enabled by language,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 2642–2646.

[8] Y. Xu, W. Li, P. Vaezipoor, S. Sanner, and E. B. Khalil, “Llms and the abstraction and reasoning corpus: Successes, failures, and the importance of object-based representations,” *arXiv preprint arXiv:2305.18354*, 2023.

[9] T. J. C. Min, “An approach to solving the abstraction and reasoning corpus (arc) challenge,” *arXiv preprint arXiv:2306.03553*, 2023.

[10] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, “Decision transformer: Reinforcement learning via sequence modeling,” *Advances in neural information processing systems*, vol. 34, pp. 15 084–15 097, 2021.

[11] J. Park, J. Im, S. Hwang, M. Lim, S. Ualibekova, S. Kim, and S. Kim, “Unraveling the arc puzzle: Mimicking human solutions with object-centric decision transformer,” *arXiv preprint arXiv:2306.08204*, 2023.

[12] S. Mirchandani, F. Xia, P. Florence, B. Ichter, D. Driess, M. G. Arenas, K. Rao, D. Sadigh, and A. Zeng, “Large language models as general pattern machines,” *arXiv preprint arXiv:2307.04721*, 2023.

[13] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar, “Voyager: An open-ended embodied agent with large language models,” *arXiv preprint arXiv:2305.16291*, 2023.

[14] X. Zhu, Y. Chen, H. Tian, C. Tao, W. Su, C. Yang, G. Huang, B. Li, L. Lu, X. Wang *et al.*, “Ghost in the minecraft: Generally capable agents for open-world enviroments via large language models with text-based knowledge and memory,” *arXiv preprint arXiv:2305.17144*, 2023.

[15] OpenAI, “Gpt-4 technical report,” 2023.

[16] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, “Retrieval-augmented generation for knowledge-intensive nlp tasks,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 9459–9474, 2020.

[17] M. Assran, Q. Duval, I. Misra, P. Bojanowski, P. Vincent, M. Rabbat, Y. LeCun, and N. Ballas, “Self-supervised learning from images with a joint-embedding predictive architecture,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 15 619–15 629.

[18] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, “High-resolution image synthesis with latent diffusion models,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 10 684–10 695.

[19] 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*, vol. 35, pp. 24 824–24 837, 2022.

[20] C. Wu, S. Yin, W. Qi, X. Wang, Z. Tang, and N. Duan, “Visual chatgpt: Talking, drawing and editing with visual foundation models,” *arXiv preprint arXiv:2303.04671*, 2023.

[21] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. Dal Lago *et al.*, “Competition-level code generation with alphacode,” *Science*, vol. 378, no. 6624, pp. 1092–1097, 2022.