Large Language Model (LLM) as a System of Multiple Expert Agents: An Approach to solve the Abstraction and Reasoning Corpus (ARC) Challenge

John Tan Chong Min [1] **Mehul Motani** [1]

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 utilise 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, we will be able to solve more.

1. 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 (Chollet, 2019). 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,

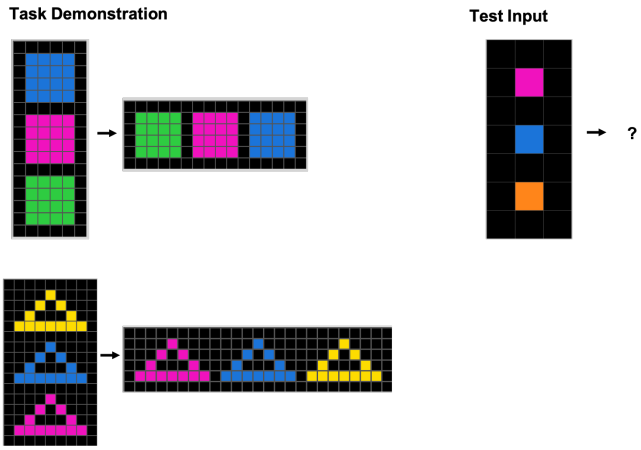

Figure 1. A sample ARC task. The challenge is to infer the abstract rule(s) governing the demonstration transformations and apply it to the test input. Example from: https://aiguide.substack.com/p/why-the-abstraction-and-reasoning

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 (Boden, 2014)) 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 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

[1]Department of Electrical and Computer Engineering, National University of Singapore. Correspondence to: John Tan Chong Min <johntancm@u.nus.edu>, Mehul Motani <motani@u.nus.edu>.

Preprint.

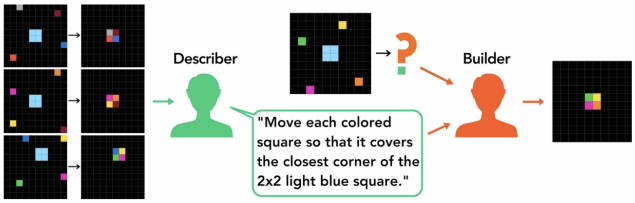

Figure 2. 88% of ARC tasks can be solved by the Builder from just the description alone given by the Describer, without input-output examples. Can GPT-4 function as both the describer and the builder? Image reproduced from Fig. 4 of Acquaviva et al. (2021).

which 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.

2. Related Work

ARC Challenge. The ARC challenge (Chollet, 2019) 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 (Alford, 2021; Ferré, 2021; Xu et al., 2023a). 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 (Camposampiero et al., 2023). Indeed, 88% of ARC tasks can be solved via language description alone without input-output examples as shown in Fig. 2 (Acquaviva et al., 2021). 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 (Xu et al., 2023b). Some work has also been done to do end-to-end input to program description generation with just LLMs alone to some success (Min, 2023). Other approaches have used Decision Transformers (Chen et al., 2021) to find a sequence of primitive actions from the input to output (Park et al., 2023), 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 (Mirchandani et al., 2023).

Code as Skills and Environmental Feedback. Voyager is an embodied lifelong learning agent powered by LLMs (Wang et al., 2023). 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 (Zhu et al., 2023) 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.

3. Broad Overview of Method

In this section, we provide an overview of our proposed approach and discuss several key ideas behind it. We have not implemented out all parts of the proposed approach, but it is already doing well. Generative Pre-trained Transformer 4 (GPT-4) is a multimodal LLM created by OpenAI and released in March 2023 (OpenAI, 2023). For now, we exclusively use GPT-4 for our model, as we empirically observe that GPT-3.5 and other open source models are not able to perform well enough for this method to work. The overall method is shown in Fig. 3.

Problem Type Classification (Not Implemented). 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

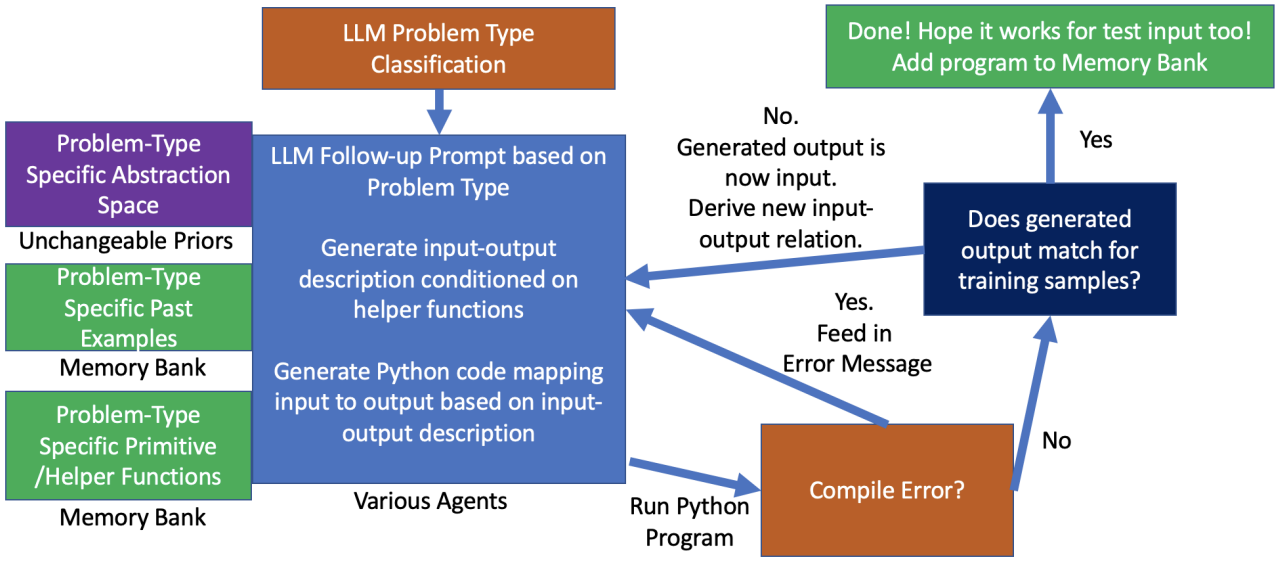

Figure 3. Process Flowchart of LLMs as a System to solve the ARC Challenge.

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 (currently) 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 (Xu et al., 2023b). 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, the agents will have to do processing based on these fixed priors.

Encoding Human Biases via Helper/Primitive Functions. An initial implementation of using GPT-4 to solve ARC was 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 (Not Implemented). 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 or fine-tuning becomes available, 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 (Lewis et al., 2020).

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 (Wang et al., 2023; Zhu et al., 2023).

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.

4. Detailed Overview of Method

We now go into some details of our method. Refer to Appendix A and B for the full GPT-4 prompt.

4.1. 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) (Assran et al., 2023) and Stable Diffusion (Rombach et al., 2022) 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. Refer to Appendix C for more details.

4.2. 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 (Wei et al., 2022), as it is done in a specific sequence to encourage broad to specific thinking.

4.3. 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 (See Appendix D for some examples of GPT output in this 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."

4.4. 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 (Wu et al., 2023), 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.

4.5. 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}

5. 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 employ a mass sampling and filtering process to generate code, much like in AlphaCode (Li et al., 2022) (see Fig. 4). *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 (See Appendix C for details). 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.

6. Results

Overall. Overall, as shown in Table 1, 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/`.

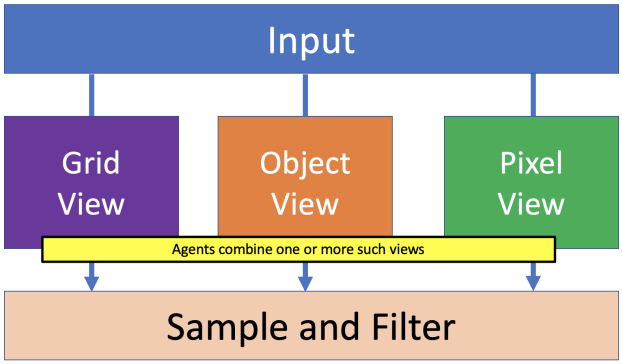

Figure 4. The overall Mass Sampling and Filtering process with various expert agents

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 2 (See Appendix E for details). 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 3 (See Appendix E for details). This is quite significant, and highlights the importance of environmental feedback.

7. 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 abstraction space, 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

Table 1. 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). See Appendix E for breakdown of tasks solved by each view type.

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

Table 2. Tasks not solved but with correct description

Total Tasks Not Solved	Correct description
61	8

Table 3. Tasks solved with iterative feedback loop after either incorrect output or compile error

Total	Incorrect Output	Compile Error
50	6	1

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.

8. 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 Xu et al. (2023b) when the performance declined after adding in relations between objects. This reinforces our idea that it is best to split up into multiple expert agents with separate views and only relevant primitive functions.

Based on our experimental results, we propose new views/agents in Appendix F.

9. 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 (Wang et al., 2023). 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.

10. Conclusion

Overall, LLMs as a system of multiple expert agents with environmental feedback is a promising approach towards solving the ARC Challenge. To facilitate further research using this approach, our code can be found at `https://github.com/tanchongmin/ARC-Challenge/`.

Acknowledgements

This research is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG-GC-2019-002). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

Many thanks for the various intelligent people who have encouraged me to pursue the GPT4 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. If your name is not here, do not worry, you can be in the future paper improving on this work:)

References

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

Alford, S. *A Neurosymbolic Approach to Abstraction and Reasoning*. PhD thesis, Massachusetts Institute of Technology, 2021.

Assran, M., Duval, Q., Misra, I., Bojanowski, P., Vincent, P., Rabbat, M., LeCun, Y., and Ballas, N. Self-supervised learning from images with a joint-embedding predictive architecture. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 15619–15629, 2023.

Boden, M. A. 4 gofai. *The Cambridge handbook of artificial intelligence*, pp. 89, 2014.

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

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., and Mordatch, I. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

Chollet, F. On the measure of intelligence. *arXiv preprint arXiv:1911.01547*, 2019.

Ferré, S. 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.

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

Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., Eccles, T., Keeling, J., Gimeno, F., Dal Lago, A., et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

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

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

OpenAI. Gpt-4 technical report, 2023.

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

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695, 2022.

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

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

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

Xu, Y., Khalil, E. B., and Sanner, S. Graphs, constraints, and search for the abstraction and reasoning corpus. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 4115–4122, 2023a.

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

Zhu, X., Chen, Y., Tian, H., Tao, C., Su, W., Yang, C., Huang, G., Li, B., Lu, L., Wang, X., 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.

APPENDIX

The appendix contains the following sections:

A Full Prompt Details for GPT-4 - This details the entire prompt used for GPT-4

B Primitive Functions and Conditional Functions - This details all the primitive functions and conditional functions used for the grounding of GPT-4

C Abstraction Views - This details the various abstraction views of Grid, Object and Pixel

D GPT-4 Output Examples - This showcases GPT-4's output to the prompt

E Task Solved Details - This details the breakdown of the tasks solved by view type

F Proposed Agent Types - This details the proposed agent types which may help increase solve rate of GPT-4 for the ARC Challenge.

A. Full Prompt Details for GPT-4

This section details the prompts used for GPT-4. The prompts are split up into a user prompt and a system prompt, as required by the GPT-4 API. If we do not use the API and use the web browser interface instead, we put the user prompt at the start of the prompt and the system prompt at the end of the prompt with the appropriate headings.

A.1. User Prompt

> All coordinates are given as (row,col). Use get_size(grid) to return (len(grid),len(grid[0])).
> To get objects, use get_objects(diag=False, by_row=False, by_col=False, by_color=False, multicolor=False, more_info=True) # replace this with whatever object view was used
>
> *[JSON with various abstraction views of input/output, and input and output grid size]*
> *[Environmental Feedback - Either empty if first iteration, otherwise either Compile Error, or Output Error Message]*

A.2. Environmental Feedback (Compile Error Message)

> Previous Code: *[Code]*
> Error Message: *[Error Message]*
> Previous Overall Pattern: *[Overall Pattern]*
> Your code had compilation errors. Correct it.

A.3. Environmental Feedback (Output Error Message)

If there is an output error (code generated output does not match Task Demonstration output), we treat this output as the new input and ask GPT-4 to get the new input-output relation.

> Use the transform_grid function to get the right relation from 'input' to 'output'

A.4. System Prompt

Refer to Appendix B for details for the helper/primitive and conditional functions.

You are given a series of inputs and output pairs.
The values from 'a' to 'j' represent different colors. '.' is a blank cell.
For example, [['.','a','.'],['.','.','b']] represents a 2 row x 3 col grid with color a at position (1,0) and color b at position (2,1).
Coordinates are 2D positions (row, col), row representing row number, col representing col number, with zero-indexing.
Input/output pairs may not reflect all possibilities, you are to infer the simplest possible relation.

[Helper/Primitive Functions Description + Example]
[Conditional Functions Description + Example]

You are to output the following in json format:
{'reflection': 'reflect on the answer',
'pixel_changes': 'describe the changes between the input and output pixels, focusing on movement or pattern changes',
'object_changes': 'describe the changes between the input and output objects, focusing on movement, object number, size, shape, position, value, cell count',
'helper_functions': 'list any relevant helper_functions for this task',
'overall_pattern': 'describe the simplest input-output relationship for all input-output pairs',
'program_instructions': 'Plan how to write the python function and what helper functions and conditions to use',
'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."}.
Do not use quotation marks ' or " within the fields unless it is required for the python code

B. Primitive Functions and Conditional Functions

This section details all the primitive functions and conditional functions used for GPT-4. Currently, these functions are not learnable, and are defined via tuning by a human over the first 50 ARC training tasks. In the future, we intend for these functions to be learned and expanded from a starting set independent of human interaction.

For more information of how these functions were implemented, refer to the Jupyter Notebook provided in https://github.com/tanchongmin/ARC-Challenge.

B.1. Primitive Functions

This is the way we prompted GPT-4 to understand the format for the primitive (helper) functions. This is essentially the

Each of the input-output relation can be done with one or more helper functions chained together.
Some relations require other functions, which you will need to come up with yourself.
Objects are tight-fitted grids (no empty row or column) with a top left coordinate, which can be used for easy manipulation of multiple coordinates.
You can create your own objects by just creating a dictionary with 'tl' and 'grid'
You can change an object's position by using 'tl' and its composition using 'grid'.
You should start each program by copying input grid or empty_grid or crop_grid of desired output size.
Then, fill the grid by using the fill helper functions.
If you use the fill functions with a '.' value, it is equivalent to removing parts of the grid.

Helper functions:
- get_objects(grid,diag=False,by_row=False,by_col=False,by_color=False,multicolor=False,more_info = True): Takes in grid, returns list of object dictionary: top-left coordinate of object ('tl'), 2D grid ('grid') by_row views splits objects by grid rows, by_col splits objects by grid columns, by_color groups each color as one object, multicolor means object can be more than one color. Empty cells within objects are represented as '$'. If more_info is True, also returns size of grid ('size'), cells in object ('cell_count'), shape of object ('shape')
- get_pixel_coords(grid): Returns a dictionary, with the keys the pixel values, values the list of coords, in sorted order from most number of pixels to least
- empty_grid(row, col): returns an empty grid of height row and width col
- crop_grid(grid, tl, br): returns cropped section from top left to bottom right of the grid
- tight_fit(grid): returns grid with all blank rows and columns removed
- combine_object(obj_1, obj_2): returns combined object from obj_1 and obj_2. if overlap, obj_2 overwrites obj_1
- rotate_clockwise(grid, degree=90): returns rotated grid clockwise by a degree of 90, 180, 270 degrees
- horizontal_flip(grid): returns a horizontal flip of the grid
- vertical_flip(grid): returns a vertical flip of the grid
- replace(grid, grid_1, grid_2): replaces all occurences of grid_1 with grid_2 in grid
- get_object_color(obj): returns color of object. if multicolor, returns first color only
- change_object_color(obj, value): changes the object color to value
- fill_object(grid, obj, align=False): fills grid with object. If align is True, makes grid same size as object
- fill_row(grid, row_num, value, start_col=0, end_col=30): fills output grid with a row of value at row_num from start_col to end_col (inclusive)
- fill_col(grid, col_num, value, start_row=0, end_row=30): fills output grid with a column of value at col_num from start_row to end_row (inclusive)
- fill_between_coords(grid, coord_1, coord_2, value): fills line between coord_1 and coord_2 with value
- fill_rect(grid,tl,br,value): fills grid from tl to br with value. useful to create rows, columns, rectangles
- fill_value(grid, pos, value): fills grid at position with value

This is the way we one-shot prompted GPT-4 for the primitive functions.

```
assert get_objects([['a','a','a'],['a','.','a'],['a','a','a']],more_info=False)==[{'tl':(0,0),grid':[['a','a','a'],['a','.','a'],['a','a','a']]} ,{'tl':(1,1),'grid':[['$']]}]
assert get_pixel_coords([['a','a'],['d','f']])=={'a':[(0, 0),(0, 1)],'d':[(1, 0)],'f':[(1, 1)]}
assert empty_grid(3, 2)==[['.','.'], ['.','.'], ['.','.']]
assert crop_grid([['a','a','b'],['.','a','b']],(0, 0),(1, 1))==[['a','a'],['.','a']]
assert tight_fit([['.','.','.'],['.','a','.'],['.','.','.']])==[['a']]
assert combine_object({'tl':(0, 0),'grid':[['a','a'],['a','.']]},{'tl': (1, 1),'grid':[['f']]})=={'tl':(0, 0),'grid':[['a','a'],['a','f']]}
assert rotate_clockwise([['a','b'],['d','e']],90)==[['d','a'],['e','b']]
assert rotate_clockwise([['a','b'],['d','e']],270)==[['b','e'],['a','d']]
assert horizontal_flip([['a','b','c'],['d','e','f']])==[['c','b','a'], ['f','e','d']]
assert vertical_flip([['a','b','c'],['d','e','f']])==[['d','e','f'],['a','b','c']]
assert replace([['a','.'],['a','a']],[['a','a']],[['c','c']])==[['a','.'],['c','c']]
assert change_object_color({'tl':(0,0),'grid':[['a','.']]},'b')=={'tl':(0,0),'grid':[['b','.']]}
assert get_object_color({'tl':(0,0),'grid':[['a','.']]})=='a'
assert fill_object([['.','.'],['.','.']],{'tl':(0, 1),'grid':[['c'],['c']]})==[['.','c'],['.','c']]
assert fill_value([['.','a'],['.','a']],(1,1),'b')==[['.','a'],['.','b']]
assert fill_row([['a','a'],['c','a']],0,'b')==[['b','b'],['c','a']]
assert fill_col([['a','a'],['c','a']],0,'b')==[['b','a'],['b','a']]
assert fill_rect([['a','a'],['c','a']],(0,0),(1,1),'b')==[['b','b'],['b','b']]
assert fill_between_coords([['.','.']],(0,0),(0,1),'a')==[['a','a']]
```

B.2. Conditional Functions

This is the way we prompted GPT-4 to understand the format for the conditional functions.

```
Each helper function can be conditional.
The conditions can be:
- by attribute, such as shape, color, position, size, cell number of object
- the condition can be an attribute on all objects, for instance, objects with the most common or least common value, or objects with the most or least common shape
- by position of pixels, such as row or column
- by neighbouring cell types or values


There are some conditional functions to help you.
- object_contains_color(obj, value): returns True/False if object contains a certain value
- on_same_line(coord_1, coord_2): Returns True/False if coord_1 is on the same line as coord_2. line_type can be one of ['row', 'col', 'diag']
```

This is the way we one-shot prompted GPT-4 for the conditional functions.

```
assert object_contains_color({'tl':(0,0),'grid':[['a']]},'a')==True
assert on_same_line((1,1),(1,2),'row')==True
assert on_same_line((1,1),(2,1),'col')==True
assert on_same_line((1,1),(2,2),'diag')==True
```

C. Abstraction Views

This section details how each abstraction view is represented.

Grid View - This provides the entire grid in ASCII format, in a 2D array. This is used when there are arbitrary patterns to find out. Note that we replace the initial JSON task input of grids with 0-9, with grids of '.','a','b','c','d','e','f','g','h','i'. This not only serves to prevent GPT-4 from trying to perform arithmetic on pixel values, but also eradicates the possibility that GPT-4 has seen the public training or evaluation set online, because it is of a different form.

Object View - This provides GPT-4 with an input view that groups contiguous cells of the same or different colour (non-zero cells) together and gives their attributes - top left coordinate, tight-fitted grid layout, shape, size, cell count. This is used when we would like to manipulate groups of related cells as one object.

Within object view, there are a few different types of categorising into objects. This is loosely inspired from Michael Hodel's DSL for ARC from `https://github.com/michaelhodel/arc-dsl`. We also have an option "more_info", which helps with reducing context length when set to False. All in all, there are 10 different kinds of Object Views.

1. Attribute 1 - Colour (2 possibilities): Mono Colour vs Multi Colour - One way is to form objects is by grouping contiguous cells of the same connected either horizontally or vertically. The other way is to group contiguous cells of any colour except of colour '.' together. These two ways are termed "Mono Colour" and "Multi Colour" respectively.
2. Attribute 2 - Constrains (5 possibilities): None/Row/Column/Colour/Diagonal - Row or column is used when the pattern is confined to just the row or column. Colour is used when we group objects by colour globally. It helps a lot to split a difficult problem into a smaller confined one where GPT-4 can perform association. Diagonal treats contiguous cells as diagonal connections as well.

Pixel View - This provides GPT-4 with a view of the pixel value and all the corresponding coordinates, in a dictionary form with the pixel value as the key and the list of coordinates as the value. This is used when the input-output relation involves changing a group of pixels. This can also be used to supplement Object View. The benefit of using pixel view is that relations like shifting, adding or removing pixels at various positions are immediately obvious to GPT-4 as compared to Grid View. The downside though, is that pixel view is not robust to offsets of positions - if the starting grid is not tight fitted and does not start at (0,0), this can lead to problems with mapping.

Figure 5. A sample grid for an ARC Challenge Task - taken from Task Demonstration 1 of ARC Training Set d037b0a7

Example: For Fig. 5, the abstraction spaces will be represented in text as:

1. **Grid View:** [['.','.','f'],['.','d','f'],['c','d','f']]
2. **Object View (Mono-Color):** [{'tl':(0,2), 'grid':[['f'],['f'],['f']], 'size':(3,1), 'cell_count':3, 'shape':[['x'],['x'],['x']]}, {'tl':(1,1), 'grid':[['d'],['d']], 'size':(2,1), 'cell_count':2, 'shape':[['x'],['x']]}, {'tl':(2,0), 'grid':[['c']], 'size':(1,1), 'cell_count':1, 'shape':[['x']]}]
3. **Pixel View:** {'f':[(0,2),(1,2),(2,2)], 'd':[(1,1),(2,1)], 'c':[(2,0)]}

D. GPT-4 Output Examples

In this section, we showcase GPT-4's output for a solved problem, and the output for an unsolved problem.

D.1. GPT-4's Output for Solved Problems

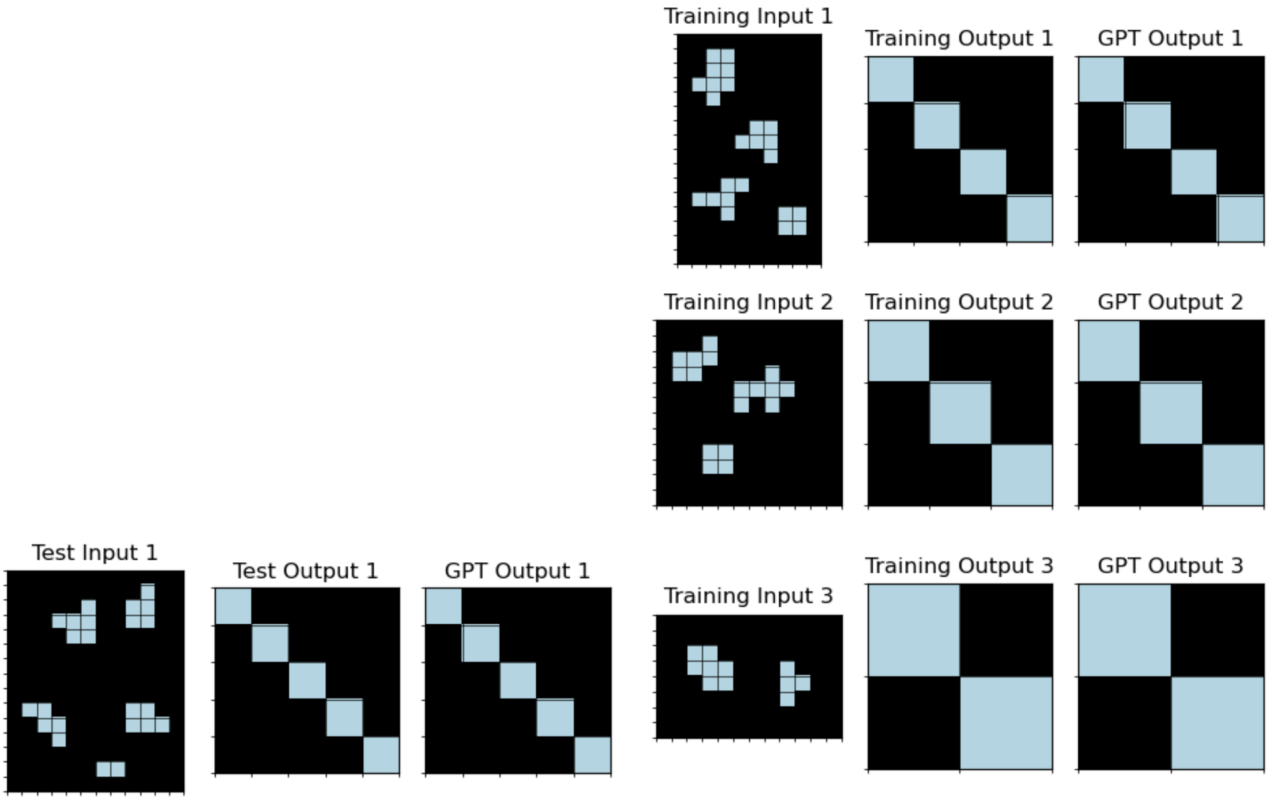

Figure 6. Training and Test Input/Output pairs for a Solved Problem, d0f5fe59, using Object View

This is the JSON output of GPT-4 formatted into key-value pairs, and the "python_program" provided correctly solves the Test Input.

reflection: This task involves objects found in the grid which are simplified and moved to a certain pattern in the output grid. The input grids contain a variety of object shapes, some of which are isolated pieces while others are complex shapes. These objects, regardless of their original form, are reduced to a single cell in the output grid and relocated to the diagonal line from the top-left to the bottom-right.

pixel_changes: In the transformation from input to output, the number of pixels with color 'h' decreases. This decrease occurs because every 'h' color object is condensed to a single pixel in the output. Also, the pixels are relocated, following a diagonal line from top left to bottom right, where the first object is placed at (0,0), the second at (1,1), and so on, regardless of their initial positions.

object_changes: The number of objects remains the same from input to output. However, their size, shape, and positions change significantly. Each object, regardless of its initial shape and size, is simplified to a single pixel in the output grid. The position of the objects follows a specific pattern in the output: they are relocated to the diagonal line from the top left to the bottom right. The sequence of placement on the diagonal line is according to the order of the objects in the input grid.

helper_functions: The relevant helper functions for this task are get_objects() for identifying objects in the input grid, fill_value() for placing single pixels at specific locations in the output grid, and empty_grid() for initializing the output grid.

overall_pattern: The task takes an input grid containing a number of objects, simplifies each object into a single cell, and repositions these cells along a diagonal line from the top left to the bottom right. The order of placement on the diagonal is according to the order of the objects in the input grid. The size of the output grid is determined by the number of objects in the input grid.

program_instructions: 1. Use get_objects() function to retrieve all objects from the input grid. 2. Create an empty grid using empty_grid(), where the number of rows and columns equals the number of objects retrieved. 3. Loop through each object obtained from get_objects(). For each object, fill a cell in the new grid with the object color at a diagonal position corresponding to the index of the object in the objects list. This is done by calling fill_value() with the index as both the row and column position.

python_program: def transform_grid(grid): objects = get_objects(grid) new_grid = empty_grid(len(objects), len(objects)) for i, obj in enumerate(objects): new_grid = fill_value(new_grid, (i, i), get_object_color(obj)) return new_grid

D.2. GPT-4's Output for Unsolved Problems

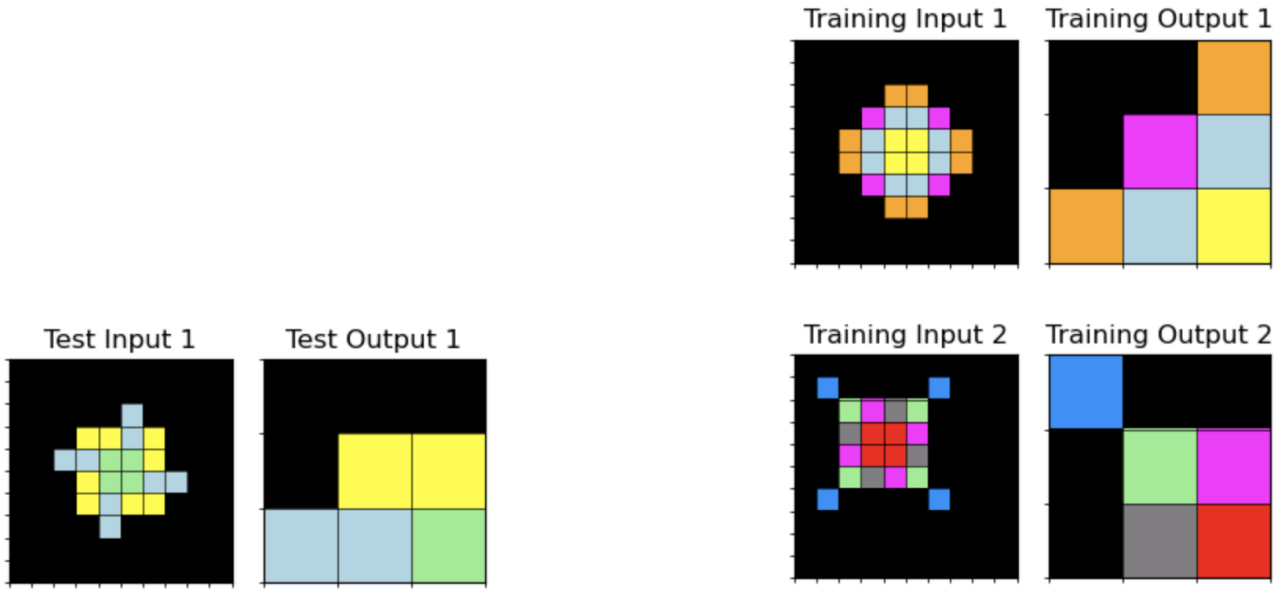

Figure 7. Training and Test Input/Output pairs for Iteration 1 for an unsolved Problem, 2013d3e2, using Object View. There was a compile error.

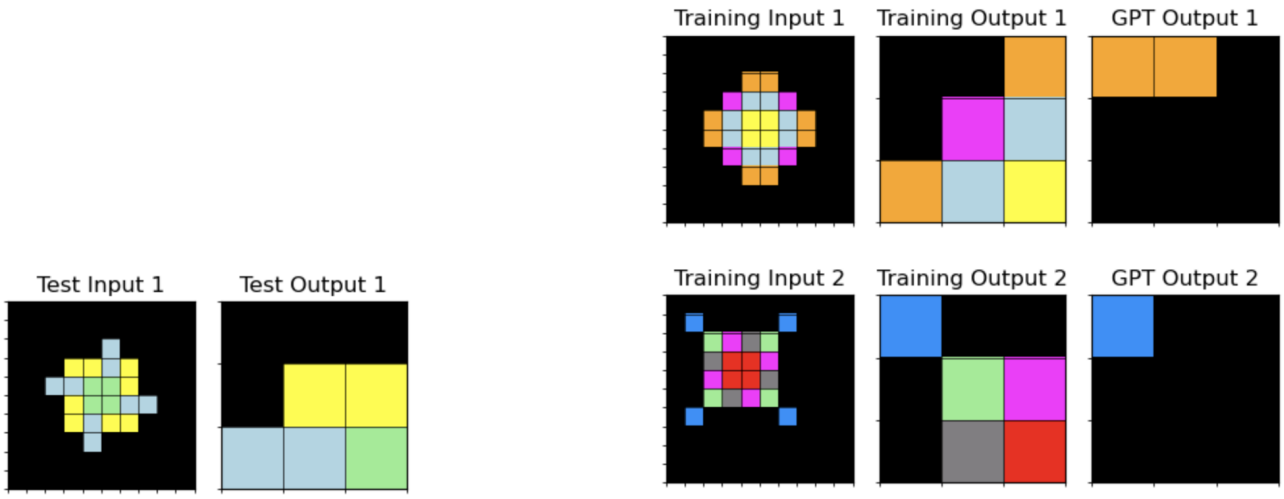

Figure 8. Training and Test Input/Output pairs for Iteration 2 for an unsolved Problem, 2013d3e2, using Object View. There was an incorrect program output. The run terminated at this iteration.

This is the JSON output of GPT-4 formatted into key-value pairs, for Iteration 1 for 2013d3e2 using Object View, and the "python_program" had a compile error. This is because it tried to crop from the object grid from the center rather than the top left of the object. It also failed to set diagonal = True and multicolor = True in the get_object settings.

reflection: The task involves identifying a specific part of an object, cropping it, and outputting it. The specific part of the object is always three rows high and starts from the third column from the left. It's interesting to note that although the objects in the inputs are of different colors and shapes, the part that needs to be cropped is always at the same location.

pixel_changes: The pixels in the output are a subset of the pixels from the input. They are always the pixels from the third, fourth, and fifth rows of the object, starting from the third column from the left. The colors of the pixels remain unchanged.

object_changes: The size of the object is significantly reduced in the output. It goes from a 6x6 square in the input to a 3x3 square in the output. The shape of the object changes as a result of the cropping, but the relative positions of the pixels within the output object remain the same as in the input object.

helper_functions: The following helper functions would be relevant for this task: get_objects, crop_grid, and fill_object.

overall_pattern: The task requires identifying the single object in the input, cropping a 3x3 square from it starting from the third column from the left, and outputting the cropped section. The part of the object that needs to be cropped is always the same and does not depend on the colors or the shape of the object.

program_instructions: 1. Start with an empty grid of size 3x3. 2. Use the get_objects function to identify the object in the input. 3. Crop a 3x3 square from the object starting from the third column from the left using the crop_grid function. 4. Use the fill_object function to fill the empty grid with the cropped section.

python_program: def transform_grid(grid): output = empty_grid(3, 3) objects = get_objects(grid) object = objects[0] cropped = crop_grid(object['grid'], (2, 2), (4, 4)) fill_object(output, {'tl': (0, 0), 'grid': cropped}) return output

This is the JSON output of GPT-4 formatted into key-value pairs, for Iteration 2 for 2013d3e2 using Object View, and the "python_program" produced an incorrect output. It managed to solve the earlier compile error by using a try-catch block and just fill the first object's grid if there is any error. The program is still wrong because it decided to crop the object grid from the center rather than from the top left corner. It also failed to set diagonal=True and multicolor = True in the get_object settings.

reflection: The task involves cropping a section from the input grid and outputting that section as the output grid. The error suggests that the cropping area defined in the function doesn't match the actual area in the object, which results in an out of range index error.

pixel_changes: The pixels in the output are a subset of the pixels in the input. A 3x3 section from the center of the only object in the input grid is extracted and used to form the output grid.

object_changes: The object in the output grid is a cropped section from the object in the input grid. There are no changes in the values of the pixels, the object simply reduces in size to a 3x3 grid. The top left cell of the output object corresponds to the cell in the third row and third column of the input object.

helper_functions: The helper functions used for this task are get_objects, crop_grid, and fill_object. get_objects is used to get the object from the input grid. crop_grid is used to crop a 3x3 section from the input object. fill_object is used to fill the output grid with the cropped object.

overall_pattern: The overall pattern is to identify the only object in the input grid, crop a 3x3 section from it starting from the third row and third column, and use this cropped section as the output grid.

program_instructions: Start by copying the input grid. Use get_objects to get the object from the input grid. Crop a 3x3 section from the input object starting from the third row and third column. Create an empty grid of size 3x3. Fill this grid with the cropped object and return it as the output. The python function will have to handle any exception to avoid out of range index error when the object is smaller than expected.

python_program: def transform_grid(grid): output = empty_grid(3, 3) objects = get_objects(grid) object = objects[0] try: cropped = crop_grid(object['grid'], (2, 2), (4, 4)) except IndexError: cropped = object['grid'] fill_object(output, 'tl': (0, 0), 'grid': cropped) return output

E. Task Solved Details

In this section, we provide a detailed breakdown of tasks solved in the main paper. For more details, check out the .html files provided in the Supplementary Materials.

Table 4 shows the breakdown of the view types used to solve the 50 problems. As we can see, a variety of views are required, of which, the most frequently used one is Object View with 23 uses. Pixel View is the next most common with 19 uses. There is 1 task (d23f8c26) which used both Object View and Pixel View. There are some tasks for which Grid View alone is the best - 7 of them. Using only one single view is bound to get poorer results than using a combination of all these views. Some examples of tasks solved by the various views are given in Figs. 10, 9, 11 and 12.

Table 4. Number of Tasks solved by each View Type. Grid View is used by default unless there is a limitation of the token length, in which case, it is toggled off

View Type	Number of Tasks Solved
Total	50
Object View	23
Pixel View	19
Object & Pixel View	1
No Object & Pixel View (only Grid View)	7

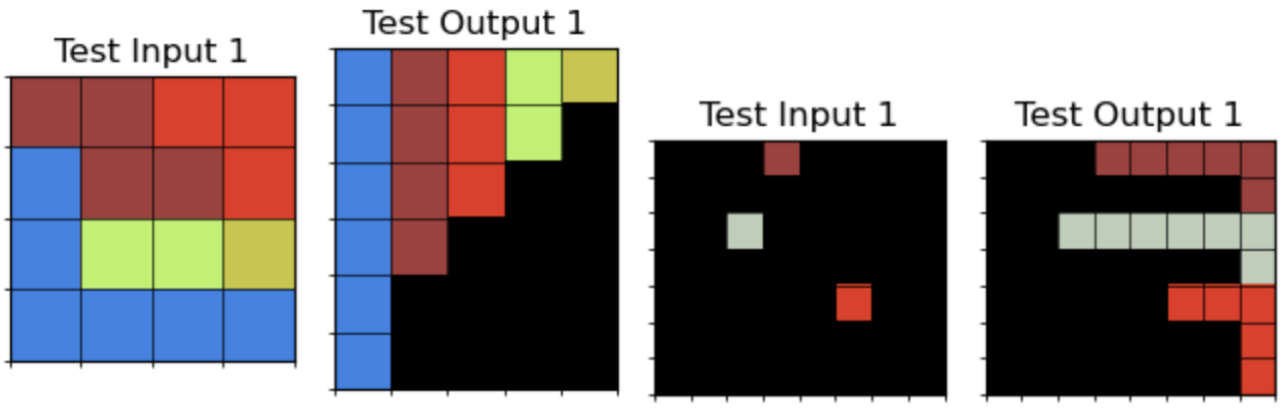

Figure 9. Examples of tasks solved using only Pixel View. They can be used for a lot of situations and usually are the only performant view when the mapping is by colour, especially for irregular shape mappings.

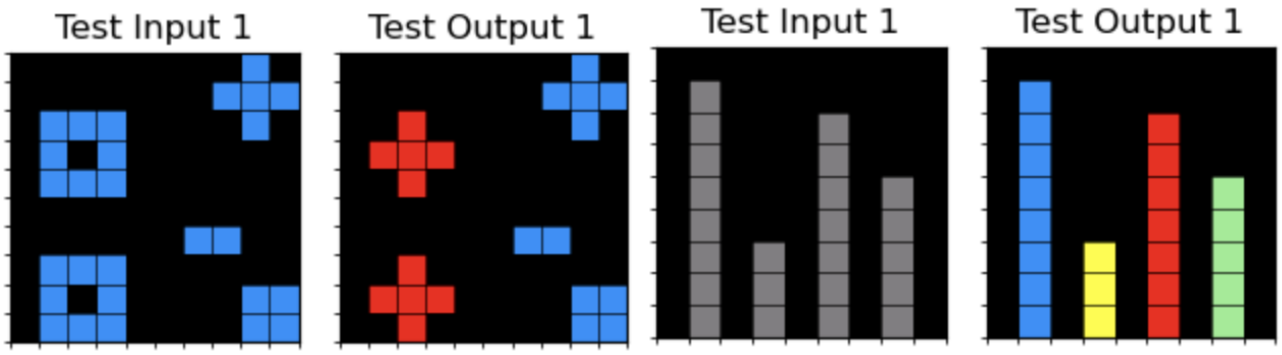

Figure 10. Examples of tasks solved using only Object View. They are usually tasks which involve changing attribute of an object (e.g. shape, size, colour) or ranking object by attributes (e.g. size, cell count).

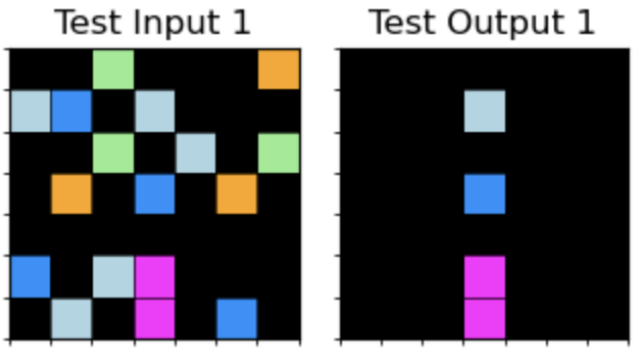

Figure 11. Examples of task solved using both Object View and Pixel View. They are usually tasks whereby we need to know the coordinates of the objects. Here, the objects are split according to columns. The task here is to just preserve the central column of pixels.

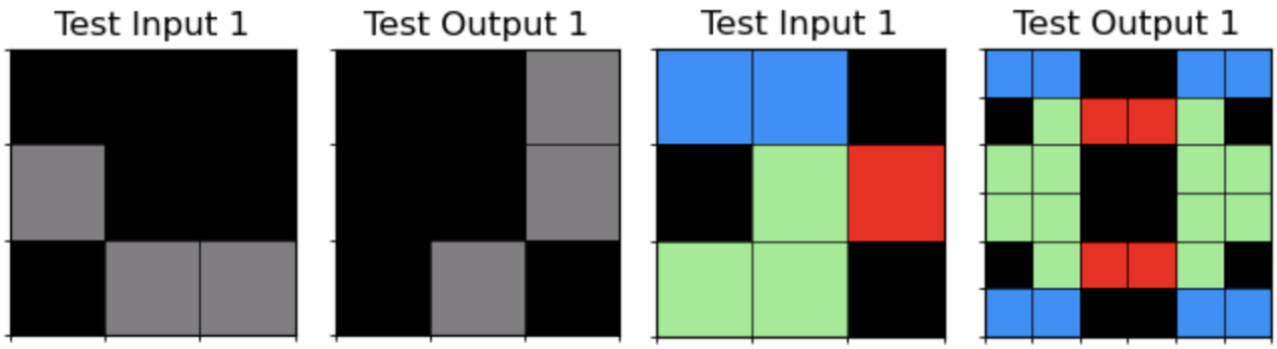

Figure 12. Examples of tasks solved using only Grid View. They are mainly rotation and reflection tasks.

Tasks Not Solved But with Overall Description Right. List of tasks which are not solved but have overall description right:

1. ea786f4a
2. 88a62173
3. 746b3537
4. 321b1fc6
5. 9565186b
6. a3df8b1e
7. f25ffba3
8. 60b61512

Tasks solved with Iterative Environment Feedback after Incorrect Output. See Fig. 13 for examples of such tasks. List of tasks which are solved over two iterated steps:

1. aabf363d
2. 496994bd
3. 3618c87e
4. 794b24be
5. 3c9b0459
6. ed36ccf7

Tasks solved with Iterative Environment Feedback after Compilation Error. See Fig. 14 for examples of such tasks. List of tasks which are solved after compilation error fixed:

1. 25d8a9c8

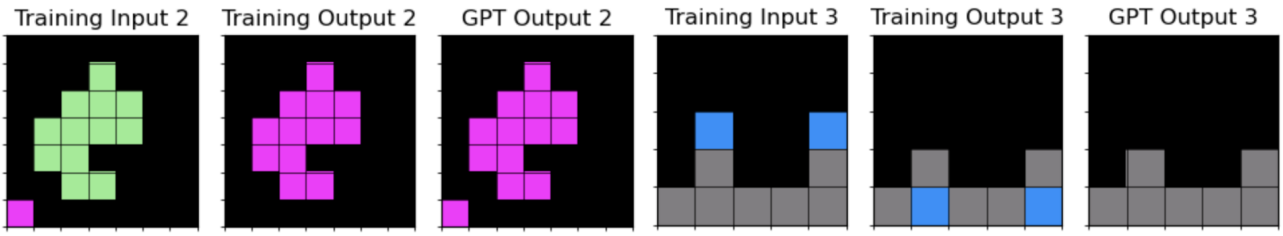

Figure 13. Examples of tasks solved using Iterative Environment Feedback after incorrect output. They are usually tasks which involve multiple different steps, for example for the left task, changing colour of one object and removing another object, and for the right task, reducing height of the column and shifting the blue pixel to the bottom of the column.

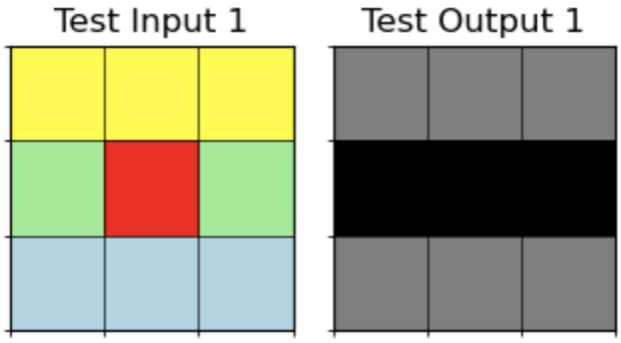

Figure 14. Examples of task solved using Iterative Environment Feedback after compile error. They are usually tasks which invoke a function that is not well explained in the prompt. Here, the program tried to do "output = copy(grid)" which is in response to the prompt "You should start each program by copying input grid or empty_grid or crop_grid of desired output size". We could eliminate this error by simply elaborating the method to use in the prompt, for example, "You should start each program by copying input grid using copy.deepcopy(grid)".

F. Proposed Agent Types

This section details more of the proposed new agent types which may help increase the solve rate of GPT-4 for the ARC Challenge.

Better Object View. We can perhaps do better by infusing more coordinates into the objects if we have context length. These coordinates would enable GPT-4 to know exactly which cell the object is positioned on. This was omitted from *Object View* because for most object-related tasks, only the top left coordinate was necessary for translation, and so by omitting this we can reduce the context length. However, having the full coordinate view could be helpful, especially when comparing the change of one object to another. This can be seen when some tasks cannot be solved by *Object View* or *Pixel View* alone, but must be solved with a combination of both *Object View* and *Pixel View*.

Object Relations View. Relations between objects is something that could be improved. Right now, GPT-4 does not have the relative positions of all objects from each other, and from the edges and corners of the grid. This may be helpful in some tasks.

Edge Detection View. We note that GPT-4 does not process shapes naturally, and is lacking some idea of what a corner of the shape is, or what is the intersection of the shape. This may be possible if we give it a view of which are the corners of the object, which are the intersections, and this can help it to understand the 2D grid better. This is akin to giving it some form of processing through a Convolutional Neural Network (CNN) type network.

Symmetry View. We observe that GPT-4 does not have a good understanding of symmetry and fails to solve most tasks involving symmetry, especially rotational symmetry. Hence, imbuing this view as well as relevant helper/primitive functions could help with this kind of problems.

Difference View. We note that finding the differences between the input and output grid is key to solving ARC tasks. Perhaps one way of grounding GPT-4 to focus on the differences between input and output apart from prompting it with text, is to explicitly calculate the differences between the input and output over various aspects, and only show GPT-4 the ones that are different. This reduces context length and could potentially help GPT-4 to attend to these differences better. This is not strictly necessary as any good enough pattern associator can already deduce what the differences and similarities are with just the raw attributes without prompting - but this could perhaps nudge the answer towards the correct one more easily. The human brain also actively seeks out differences and that is how we learn, so having something to highlight differences could perhaps improve the system better.

Diagonal View. Being a token-based predictor, GPT-4 is actually quite weak at predicting diagonals since 2D representation is not naturally imbued in tokens. In fact, we observe that GPT-4 does row prediction better than both column and diagonal prediction. As such, it would be better if we could frame any problem as a row-based problem, including diagonals. One way to do this will be to rotate the grid 45 degrees so that diagonals become rows and columns, do manipulation on this grid, and then rotate it back -45 degrees so as to get back the original input grid.