Neural Guided A* Search for the ARC-AGI Benchmark

VICKI LI*, PAWAN JAYAKUMAR*, and LUCAS DIONISOPOULOS*, UC San Diego, USA

The ARC-AGI benchmark challenges machine learning systems to solve visual reasoning tasks that require generalization beyond memorized patterns or large training datasets. This paper introduces a novel hybrid approach combining neural and symbolic methods to address the unique challenges of ARC. We attempt to use a vision transformer to encode ARC problems into dense embeddings, guiding a custom A* search over transformations defined in a domain-specific language. To represent ARC tasks more effectively, we propose an object-centric approach, where input and output grids are decomposed into ARC-objects with structured attributes. Despite having limitations, our approach solves four public evaluation cases from the ARC dataset, demonstrating an early proof-of-concept for an object-centric approach and paving the way for further experimentation to enable full neural guided search.

Additional Key Words and Phrases: Program Synthesis, Neurosymbolic Programming

1 INTRODUCTION

The ARC-AGI benchmark (ARC)[3] was created to measure machine intelligence. It is composed of visual reasoning questions that include two to seven input-output examples followed by a single test input – the goal is to correctly generate the test output. The benchmark has a goal of 85%, at which the challenge is deemed to be 'solved'. While many deep learning-based models have excelled at various other benchmarks, they continue to struggle with ARC for several reasons:

- Due to the novel nature of every problem, it is infeasible to simply memorize solutions.
- Deep learning is sample inefficient – ARC requires the ability to learn from a small handful of examples.
- The answer must be entirely correct requiring full precision.

These challenges highlight the difference between excelling at one trained task versus composing existing priors to solve unseen problems. The former is skill, the latter is intelligence.

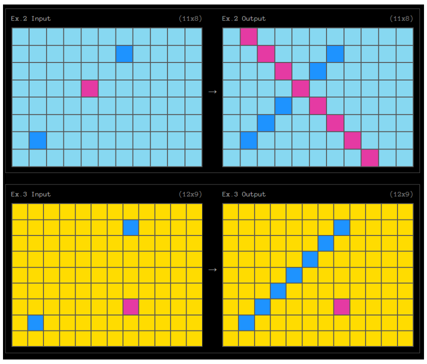


Fig. 1. Example of an ARC problem. The transformation requires connecting blue dots with a blue line and drawing a perpendicular pink line if there is a pink dot in the way.

Inspired by AlphaGo[15], we seek to combine deep learning methods with search, specifically program synthesis. Our approach segments broader images into 'ARC objects', which will be encoded

*Authors contributed equally to the paper

Authors' address: Vicki Li, v8li@ucsd.edu; Pawan Jayakumar, pjayakumar@ucsd.edu; Lucas Dionisopoulos, ldionisopoulos@ucsd.edu, UC San Diego, USA.

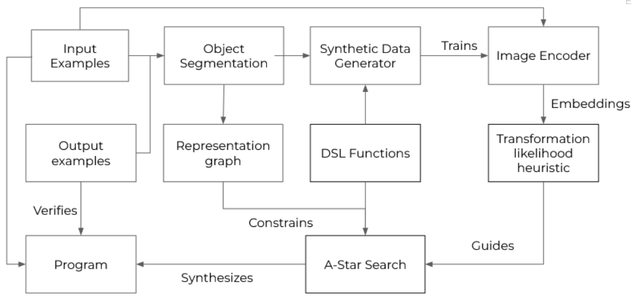
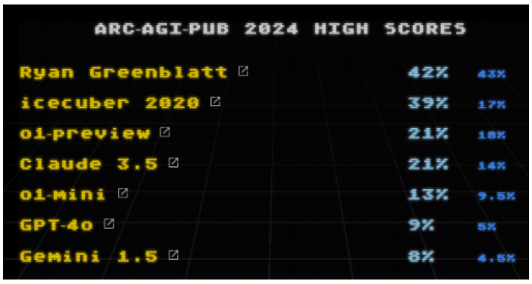


Fig. 2. Diagram of our proposed method.



ARC-AGI-PUB 2024 HIGH SCORES

Ryan Greenblatt	42%	43%
icecuber 2020	39%	17%
o1-preview	21%	18%
Claude 3.5	21%	14%
o1-mini	13%	9.5%
GPT-4o	9%	5%
Gemini 1.5	8%	4.5%

Fig. 3. Results of current enterprise LLMs on ARC. Note that `icecuber 2020` has a large gap between public and private eval, highlighting the difficulty of generalizing DSL based approaches.

as dense embedding vectors. These will be used to compute a distribution over a domain-specific language (DSL) that defines our program space, which we will then search over using A* search. While promising, our implementation has limitations due to the difficulty of learning a rich embedding model. However, our solution is still able to solve 4/400 public validation test cases, supplying a proof-of-concept to our object-centric approach that encourages further experimentation.

2 RELATED WORK

Creating a DSL to describe the structure of ARC solutions[9][16] gives search methods an inductive bias toward what transformations are likely. However, it is non-trivial to design an effective DSL that enables shallow depth solutions to all ARC problems. The immense problem novelty and combinatorial explosion of search presents a significant hurdle to the inductive program synthesis technique.

Current state-of-the-art solutions leverage pre-trained LLMs, and two common techniques have recently demonstrated efficacy. First, some approaches seek to fine-tune the LLM at test time on ARC problems[1]. By leveraging synthetic data, there is ample data to perform test-time-training – which is a powerful alternative to existing techniques such as chain-of-thought sampling[17]. The second common LLM approach requires significant sampling of LLMs for programs[7], typically in Python. The LLMs generate programs until a valid solution is generated or a compute limit is reached – yielding strong results but significant costs.

While ARC problems are ultimately reduced to JSON files, they are better presented to LLMs in other ways. Humans view these problems as visual transformations and invoke ideas such as translation, symmetry, or color. Other approaches[12] propose adding object positional embeddings to embed the concept of objectness into neural representations of ARC problems. Another technique adopted a similar object-centric method, although it did not use vision models[6].

Ultimately, none of these works have solved the core challenge of ARC which is automating skill acquisition. Each requires expert domain knowledge to be incorporated into the solution, producing bespoke solutions that are less likely to transfer to more open-ended or challenging domains such as automated theorem proving.

3 DSL AND SEGMENTATION

3.1 DSL

Our DSL has nine operations: Color, recolor, rotate, flip, delete, translate, single copy, copy translate, and draw line. While being relatively concise, these operations were designed to be general and effective when applied at the object level. Additionally, a small DSL creates a more tractable program synthesis problem, which further motivated concision.

3.2 ARC-Objects

Our approach treats ARC problems not as a pair of input-output grids, but instead as input and output ARC-object sets. Our custom ARC-object data structure consists of the mask of the pixels which comprise the object, the position (top-left corner of the bounding box of the mask) of the object in the original grid, and the height and width of the object.

An object-based method provides several advantages:

- **Simplicity**: It is easier to describe transformations on objects rather than specific pixels on a grid.
- **Modularity**: All DSL operations are applied on an ARC-object, with some requiring additional constants, and return the transformed ARC-object. This allows us to chain operations and ensure our program remains well-typed.
- **Feature Abstraction**: An object-centric approach makes it easy to extract meaningful features such as color, shape, and position.

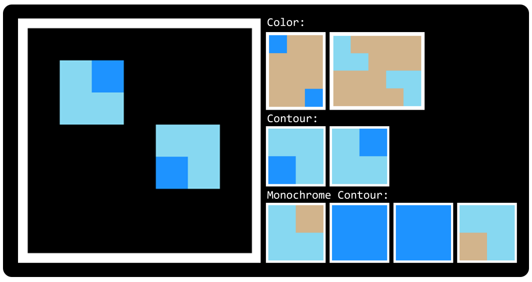


Fig. 4. Different segmentation results on same input image

3.3 Object Segmentation

We defined three methods to segment the input grids into objects, utilizing the OpenCV library to detect contour lines in a grid. The differences can be seen in Figure 4.

(1) **Color**: Treat all pixels of the same color in a grid as one object. The pixels do not have to be connected. Thus, an object can contain several disconnected parts.

(2) **Contour-Scale**: First convert the ARC grid into gray scale, scaling pixel values with respect to ascending pixel frequency. This generalizes the concept of a 'background color' and extracts distinct objects (potentially multi-colored) that are separated in the grid.
(3) **Monochrome Contour**: Combines the constraints of color and contour. All pixels in an object must be connected and of the same color.

In practice we execute all three in parallel as each is effective in various settings. Further work may provide an effective way to choose one segmentation method.

4 ENCODING

In order to generate effective heuristics for our search method, we needed some way to produce a numerical representation of the current problem state. Two common approaches to training meaningful embeddings are to use hand-designed embeddings or to use a learned embedding model. We chose the latter approach due to task complexity and for generality.

We had hoped a performative open-source embedding model such as ResNet[8] and MobileNet[13] could have provided embeddings, but common practice in training image classification models involves using data augmentations to impose robustness against color, rotation, size, and position within the image[2]. This is not ideal as we want our encoder to be sensitive towards these characteristics. Additionally, these models are trained on common image classification tasks that are different from ARC challenge problems, and the large internal dimensionality of these models imposes significant computational constraints for downstream tasks.

For these reasons, we chose to pre-train a task-specific ARC-object embedding model that converts an ARC-object into a semantically meaningful vector.

4.1 Architecture

We leveraged the vision transformer architecture with a *class* token to extract embeddings[4]. For our objective function, we trained using a modified iBOT objective that leveraged a student network and a teacher network with a small masking probability ($p = 0.1$) [18]. Inputs were provided as a batch of problems (each image padded to *32 x 32*) and the model was tasked with minimizing cross-entropy loss comparing the *class* tokens from the student network to the teacher network.

We used an embedding dimension of 64, 2-D sinusoidal positional encodings, 8 attention heads, transformer depth of 10, and an MLP hidden ratio of 128, which resulted in approximately $400k$ parameters. The model was trained over 4 hours on an NVIDIA A6000 GPU.

4.2 Synthetic Data

To facilitate learning relevant representations, we created a pipeline to generate synthetic data using our DSL. Many state-of-the-art ARC solvers utilize synthetic data generators such as RE-ARC[10] for the ability to extend the limited training set size. Using a synthetic data generator not only allows us to generate significantly more training data than is provided, but it also allows us to bias our model toward learning representations that distinguish our specific DSL transformations.

Examples of our synthetic data generator and scaling behavior are provided in Appendix A. We exhibited log-linear loss improvement when using our synthetic data. Notably, we were able to train on $20k$ training steps (approximately $200k$ samples) – which would have required around 100 epochs if using the provided ARC training samples. This would have posed a much higher risk of over-fitting compared to our synthetic generation regime, where we only needed 20 epoch to generate this number of samples. Additionally, the log-linear improvement in loss exhibited adds credibility that our synthetic data generation is generating useful samples – allowing the model to continue to improve on the task.

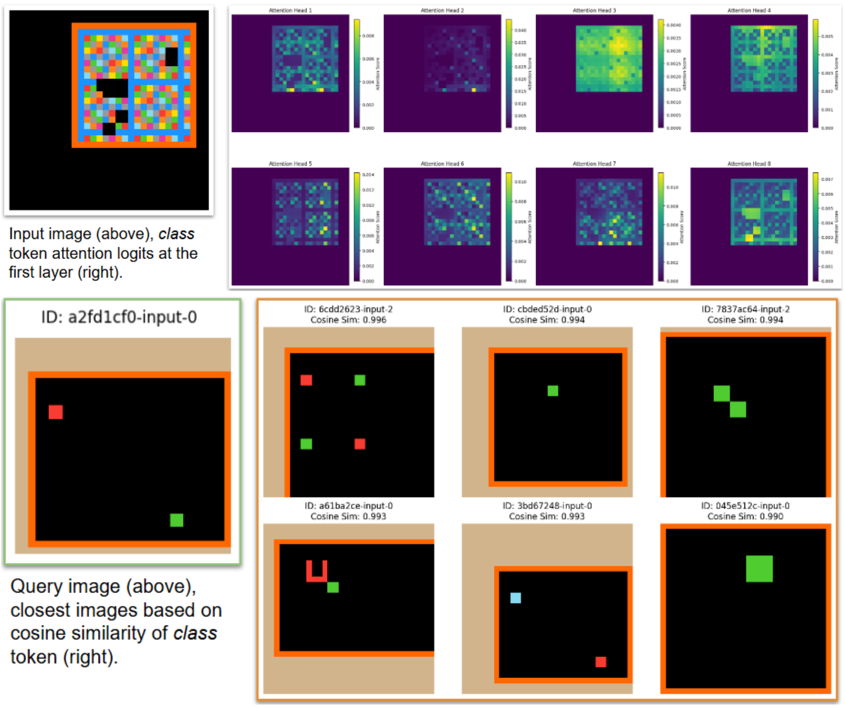


Fig. 5. (a) *Top*: Plot of attention logits for the *class* token in the first transformer layer across the 8 attention heads. (b) *Bottom*: The result of cosine similarity search over 1,000 randomly sampled ARC grids.

4.3 Cosine Similarity Evaluation

As an intermediate evaluation for our representation model, we conducted qualitative tests using cosine similarity and visualization of our attention mechanism. Several methods were tried to quantify this performance, but we believed they were ineffective at separating signal from noise.

Figure 5 displays the results from our cosine similarity evaluation. These results showed that the image encoder was learning useful representations – able to discern elements such as sparsity, color, and positioning.

4.4 Training a Probabilistic Grammar and A-Star

Given the combinatorial challenge of program synthesis, we need to train a probabilistic grammar to allow our search to be biased toward useful solutions – taking an approach similar to Euphony[11]. To do so, we proposed using an encoder-based model with two task-specific tokens. Appendix B includes an overview of the proposed architecture.

However, in doing so, we found that the model could only learn trivial representations that matched the underlying distribution. This could be due to several different reasons, but most likely it is caused by our embeddings lacking a rich representation to allow the model to learn a nontrivial distribution given the state. This remains an open problem that is a promising subject for future work.

5 OBJECT GRAPH

5.1 Motivation

Several of our DSL functions require extra arguments beyond the object itself. For example, translate requires the end coordinates of the translation. While enumerating possible arguments, it is important that 1) they generalize to other test examples and 2) do not use information specific to the output grid for a particular example as this information is not available for the test example. Effectively, finding these arguments is its own synthesis problem, which we solve using constraints. We make the assumption that these arguments are some function of the properties and relations between input objects and output objects.

5.2 Construction

Since ARC problems decompose into a set of input and output objects, we can create a graph where the nodes are the objects and edges are properties two objects share such as shape or color. We observe that many ARC problems have input objects that are modified but can still be recognized in the output grid based on some invariant property. For example in Figure 6 we observe that the shape and color of each object does not change. We can use this information to create a mapping between the set of input and output objects. This mapping can also be constructed between input objects across examples. Using these mappings, we can determine what properties are important, and further refine our search to figure out the exact expression. For example in Figure 6 we observe that only the y-coordinate of each object changes, and that they ultimately match the y-coordinate of the blue object.

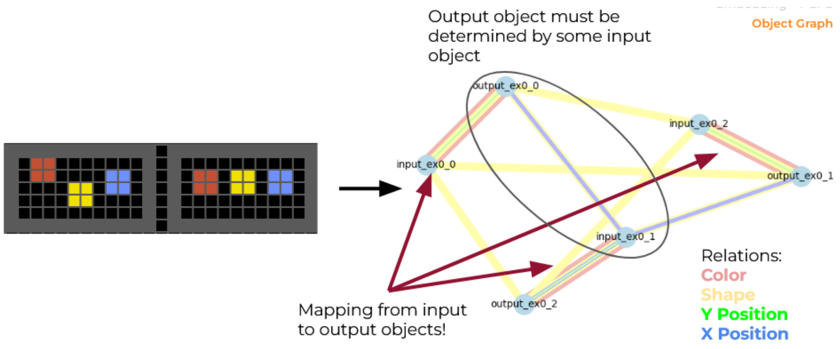


Fig. 6. Classic translation problem: The system must determine that the red and yellow objects must be translated to match the y-coordinate of the blue object.

6 RESULTS

We managed to solve 16 of 400 train problems and 4 of 400 evaluation problems in the ARC Prize dataset, using a simplified version of the object representation graph. The simplified version relieves most of the constraints on the strict input to output object mapping, and instead only focuses on the generalized feature comparison between all input objects and all output objects. Training and evaluation does not apply for this method, hence these can be simply interpreted as easy and hard problems: The training set typically contains problems that can be solved in as few as one or two transformations, while evaluation problems require longer chains of operations and much more complicated feature extraction. Examples of solved problems can be found in Appendix C.

7 CONCLUSION AND FURTHER WORK

While the initial goal of tackling the ARC-AGI benchmark with a neural-guided program synthesis approach over the object domain was unsuccessful, this has opened several promising directions for further research.

The ability to solve ARC problems using our object-level DSL and the object graph has provided a proof-of-concept that this technique can be effective. The DSL was specifically designed to allow us to solve the majority of ARC problems when applied properly on the correct ARC object.

From here, there are several improvements that could be made in all elements of the system. What remains most promising is full neural-guided search, ideally proving our belief that you can effective generalize ARC solutions to new tasks by a neurosymbolic approach. From here, further work could seek to allow the system to generate its own DSL, bootstrapping the embedding model and probabilistic grammar with each new addition – similar to the approach in DreamCoder[5]. Further, taking inspiration from MuZero [14], building a full end-to-end system to allow for an 'agential' approach to solving tasks with minimal external imposed bias remains a grand challenge that could allow an ARC solution to generalize to open-ended domains.

REFERENCES

[1] Ekin Akyürek, Mehul Damani, Linlu Qiu, Han Guo, Yoon Kim, and Jacob Andreas. 2024. The Surprising Effectiveness of Test-Time Training for Abstract Reasoning. *ArXiv* (2024). https://arxiv.org/abs/2411.07279

[2] Randall Balestriero, Mark Ibrahim, Vlad Sobal, Ari Morcos, Shashank Shekhar, Tom Goldstein, Florian Bordes, Adrien Bardes, Gregoire Mialon, Yuandong Tian, Avi Schwarzschild, Andrew Gordon Wilson, Jonas Geiping, Quentin Garrido, Pierre Fernandez, Amir Bar, Hamed Pirsiavash, Yann LeCun, and Micah Goldblum. 2023. A Cookbook of Self-Supervised Learning. arXiv:2304.12210 [cs.LG] https://arxiv.org/abs/2304.12210

[3] François Chollet. 2019. On the Measure of Inteligence. *ArXiv* (2019). https://arxiv.org/abs/1911.01547

[4] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. arXiv:2010.11929 [cs.CV] https://arxiv.org/abs/2010.11929

[5] Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Lucas Morales, Luke Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B Tenenbaum. 2021. Dreamcoder: Bootstrapping inductive program synthesis with wake-sleep library learning. In *Proceedings of the 42nd acm sigplan international conference on programming language design and implementation*. 835–850.

[6] Sébastien Ferré. 2023. Tackling the Abstraction and Reasoning Corpus (ARC) with Object-centric Models and the MDL Principle. arXiv:2311.00545 https://arxiv.org/abs/2311.00545

[7] Ryan Greenblatt. 2024.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385 [cs.CV] https://arxiv.org/abs/1512.03385

[9] Michael Hodel. 2023. ARC DSL. https://github.com/michaelhodel/arc-dsl

[10] Michael Hodel. 2024. RE-ARC: Reverse-Engineering the Abstraction and Reasoning Corpus. https://github.com/michaelhodel/re-arc

[11] Woosuk Lee, Kihong Heo, Rajeev Alur, and Mayur Naik. 2018. Accelerating search-based program synthesis using learned probabilistic models. *ACM SIGPLAN Notices* 53, 4 (2018), 436–449.

[12] Wenhao Li, Yudong Xu, Scott Sanner, and Elias Boutros Khalil. 2024. Tackling the Abstraction and Reasoning Corpus with Vision Transformers: the Importance of 2D Representation, Positions, and Objects. *ArXiv* (2024). https://arxiv.org/abs/2410.06405

[13] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. https://arxiv.org/pdf/1801.04381v4

[14] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy P. Lillicrap, and David Silver. 2019. Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. *CoRR* abs/1911.08265 (2019). arXiv:1911.08265 http://arxiv.org/abs/1911.08265

[15] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. 2017.

Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *CoRR* abs/1712.01815 (2017). arXiv:1712.01815 http://arxiv.org/abs/1712.01815

[16] Top-Quarks. 2020. ARC-Solution. https://github.com/top-quarks/ARC-solution

[17] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *NeurIPS* (2022). https://arxiv.org/abs/2201.11903

[18] Jinghao Zhou, Chen Wei, Huiyu Wang, Wei Shen, Cihang Xie, Alan Yuille, and Tao Kong. 2022. iBOT: Image BERT Pre-Training with Online Tokenizer. arXiv:2111.07832 [cs.CV] https://arxiv.org/abs/2111.07832

APPENDIX

A SYNTHETIC DATA GENERATION EXAMPLES

Samples from the synthetic data generator using a program depth of 2. Program depth is an adjustable argument – this allowed the use of curriculum learning during training to train on increasingly difficult (i.e., shallower) transformations.

At each step, we choose to apply our transformation at the image level or at the object level (via random choice). Transformations available at the image level include recolor, rotation, and flip; at the object level, we allow for these along with translation, copying, and draw-line.

Throughout the synthetic generation, objects are tested for observational equivalence with past generations. For example, generating 'rotate' four times would result in a trivial transformation at depth 4. When an observational equivalence is detected, the intermediate transformations are discarded and the synthetic generation is continued until the desired depth is reached.

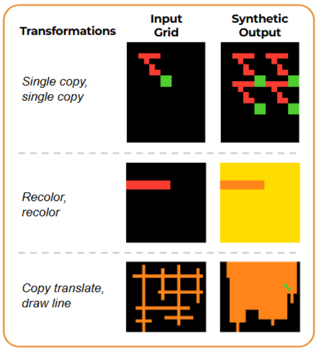


Fig. 7. Synthetic data examples at *depth* = 2.

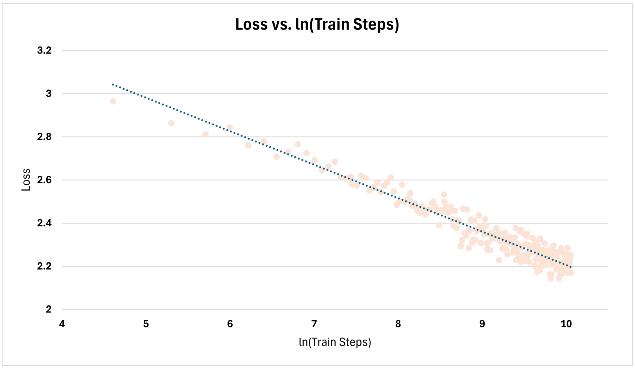


Fig. 8. Training loss using synthetic data displays a log-linear relationship with respect to training steps, highlighting that the synthetic data being generated is useful for continued improvement on the task.

B PROBABILISTIC GRAMMAR ENCODER

The attempted architecture for our probabilistic grammar encoder sought to leverage an encoder-only transformer model with two task-specific tokens. The first token would be used to predict the DSL function given the current state and desired goal state. After applying softmax on this prediction, we would have a probability distribution over our grammar that we could use for our search algorithm.

The second task token would not be directly used, but rather the attention in the last layer of the encoder would be used to generate a probability distribution over which ARC objects to apply our DSL function to. We believed that in order to predict the correct DSL function, there must also be information across

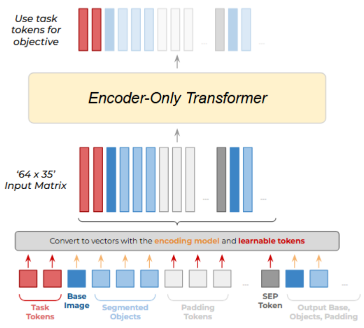

Use task tokens for objective
Encoder-Only Transformer
'64 x 35' Input Matrix
Convert to vectors with the encoding model and learnable tokens
Task Tokens
Base Image
Segmented Objects
Padding Tokens
SEP Token
Output Base, Objects, Padding

Fig. 9. Attempted architecture for the probabilistic grammar.

which object to apply that function to – this was our attempt at capturing that information.

Unfortunately, neither task was successful in tandem or on its own. This is an area for further study.

C SOLVED ARC TASKS

Figure 10 shows the four evaluation problems we solved. These problems are of moderate complexity, and the solutions involve a chain of operations. Taking the problem on the top-right corner as example, the final solution our program produced is to first segment the grid by color, flip each object, and finally flatten the list back to one grid, drawing the transformed objects to their original positions.

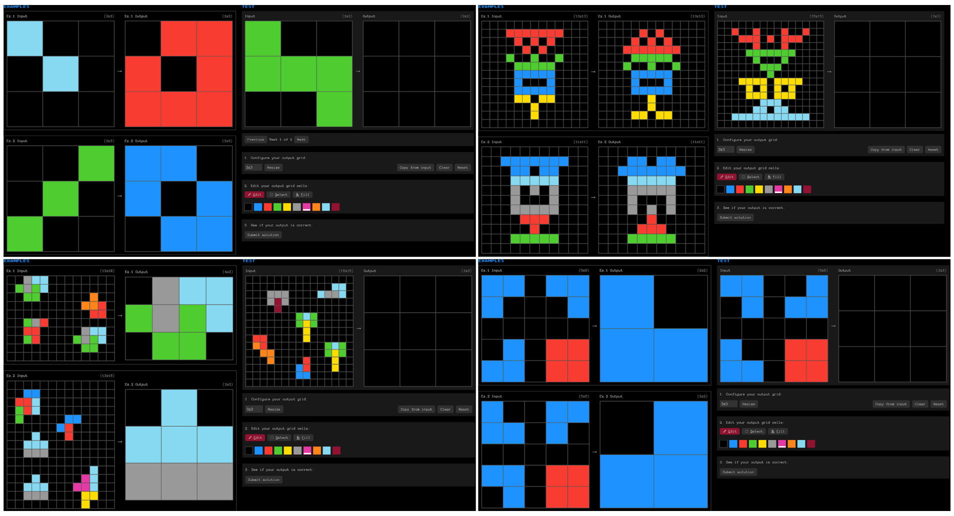


Fig. 10. ARC problems solved in the evaluation set.