
ARC-GEN: A Mimetic Procedural Benchmark Generator for the Abstraction and Reasoning Corpus

Michael D. Moffitt

Google

moffitt@google.com

Abstract

The Abstraction and Reasoning Corpus remains one of the most compelling and challenging benchmarks for tracking progress toward achieving Artificial General Intelligence. In contrast to other evaluation datasets designed to assess an agent’s task-specific skills or accumulated knowledge, the ARC-AGI suite is specifically targeted at measuring *skill acquisition efficiency*, a trait that has (so far) been lacking in even the most sophisticated machine learning systems. For algorithms that require extensive intra-task exemplars, a significant constraint imposed by ARC-AGI is the modest cardinality of its demonstration set, comprising a small number of $\langle \text{input}, \text{output} \rangle$ grids per task specifying the corresponding transformation. To embellish the space of viable sample pairs, this paper introduces ARC-GEN, an open-source procedural generator aimed at extending the original ARC-AGI training dataset as faithfully as possible. Unlike prior efforts, our generator is both *exhaustive* (covering all four-hundred tasks) and *mimetic* (more closely honoring the distributional properties and characteristics embodied in the initial ARC-AGI-1 release). We also discuss the use of this generator in establishing a static benchmark suite to verify the correctness of programs submitted to the *2025 Google Code Golf Championship*.

1 Introduction

The rapid advance of deep learning systems has been accompanied by an incredible explosion in benchmarks used to facilitate their training and subsequent evaluation. While Large Language Models require inputs tailored to conversation [1, 2] and question answering [3], other systems are designed to ingest data from a wide range of modalities, such as images [4, 5], spoken language [6, 7], and long-form videos [8]. Recent improvements in special purpose models have also given rise to datasets devoted to certain areas of subject matter expertise, including protein folding [9], code generation & refactoring [10–12], and modern mathematics [13]. Many benchmark suites place a considerable emphasis on *knowledge mastery*, containing problems whose solutions often require the expenditure of hours worth of effort on behalf of human experts. The difficulty of such tasks is perhaps best exemplified by Humanity’s Last Exam [14], a closed-ended academic benchmark intended to span the absolute frontier of human knowledge.

For a variety of reasons, one benchmark set in particular – the *Abstraction and Reasoning Corpus* [15] – stands apart from the rest. First, the dataset is remarkably compact; i.e., the entire suite of training instances requires only 1.44MB of storage (uncompressed), small enough to fit onto a 3.5" floppy disk.¹ Second, the problems are easily approachable (if not trivial) for most humans, with the vast majority capable of being solved by non-experts [17] and even children [18]. Third, the benchmark has proven somewhat resistant to deep learning techniques, as partly evidenced by the (unclaimed) top prize offered for a sufficiently high-scoring submission to the \$1 million ARC Prize 2024 challenge [19, 20].

¹By comparison, the ImageNet Object Localization dataset [16] is 167.62 GB, roughly 100,000× larger.

One key characteristic of this dataset (hereafter referred to as ARC-AGI-1)² is the limited cardinality of samples per task, which are represented as two-dimensional arrays of digits. Since the ARC-AGI format embodies transformations in a way that is highly differentiated from other problem representations available on the internet, this lack of examples can pose a challenge when training a model how to identify the appropriate mapping for any given task. Likewise, the risk of contamination grows significantly as ARC-related tasks increase in popularity, thus complicating the fair evaluation of ARC solvers. These issues have led to the development of various ARC-themed benchmark generators, yet such systems are not designed (and therefore certainly not guaranteed) to represent all original puzzles exactly, or ensure compatibility with programs that have been developed to solve them [23, 24].

In this paper, we present an open-source procedural benchmark generator named ARC-GEN aimed at extending the original ARC-AGI training dataset as faithfully as possible. Unlike previous efforts, our generator is both *exhaustive* (covering all four-hundred tasks) and *mimetic* (honoring distributional properties and characteristics similar to those embodied in the initial ARC-AGI-1 release). We demonstrate these attributes empirically by validating its ability to reproduce all examples in the original benchmark suite, and by verifying its consistency with respect to programs designed to implement each ARC-AGI transformation. Finally, we discuss an application of this generator in establishing a test harness for the *2025 Google Code Golf Championship*, a competition that requires a large number of examples to certify the correctness and generality of submitted programs.

2 Background

We begin by reviewing some basic concepts surrounding abstraction and reasoning, along with their manifestation in the ARC benchmark suite. Our terminology is consistent with prior works studying various measures of intelligence [15], albeit truncated for the sake of expositional brevity.

2.1 Algorithmic Information Theory

Consider a list of tasks $\langle \mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_n \rangle$ where each task \mathcal{T}_i performs some arbitrary predefined state transformation:

$$\mathcal{T}_i : \mathcal{S}_i \rightarrow \mathcal{S}'_i \quad \forall i \in [1, n]$$

In addition, suppose a finite set of examples $\langle \mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n \rangle$ such that each example in \mathcal{P}_i is consistent with the corresponding transformation \mathcal{T}_i :

$$\mathcal{P}_i : \langle (s_{i,1}, s'_{i,1}), (s_{i,2}, s'_{i,2}), \dots \rangle \mid s_{i,j} \in \mathcal{S}_i, s'_{i,j} = \mathcal{T}_i(s_{i,j})$$

The meta-task of *skill acquisition* is to infer \mathcal{T}_i from the elements of \mathcal{P}_i alone such that for any new state $\hat{s}_{i,j} \in \mathcal{S}_i$, the expected output $\hat{s}'_{i,j} = \mathcal{T}_i(\hat{s}_{i,j})$ can be produced. This process requires *generalization*, and the intelligence of a system can be argued to correlate directly with its ability to perform this generalization *efficiently*.

2.2 The Abstraction and Reasoning Corpus

First introduced in 2019, the Abstraction and Reasoning Corpus [15] considers a variety of tasks involving the transformation of two-dimensional grids. Such puzzles usually have a clear visual or geometric interpretation (with common themes including *translation*, *rotation*, *dilation*, *submatrix selection*, etc.), yet no two tasks are exactly the same. Most lend themselves to concise natural language descriptions [25]; refer to Figure 1 for an English translation of one puzzle (ID: 543a7ed5).

The number of examples varies by problem, but typically ranges between three and five per task. For illustration, we show in Figure 2 the only $\langle \text{input}, \text{output} \rangle$ pairs provided for puzzle 543a7ed5. This relatively small number of examples is reasonably justified by an expectation that any competent agent should need only a handful such instances to produce the desired transformation outputs, and (thankfully) this few-shot behavior has commonly been observed in practice [26, 27]. A secondary rationale is the enormous amount of manual work that went into producing & verifying this set, as all examples were painstakingly constructed by hand over the course of several months.

²A successor to this set (named ARC-AGI-2) was released on March 24th, 2025 for ARC Prize 2025 [21, 22].

[INPUT] Three non-overlapping pink rectangles (potentially hollowed out) placed upon a 15x15 cyan background.

[OUTPUT] Those same three pink rectangles each surrounded by a 1-pixel green border and all holes shaded yellow.

Figure 1: A natural language description of an ARC-AGI-1 puzzle (ID: 543a7ed5).

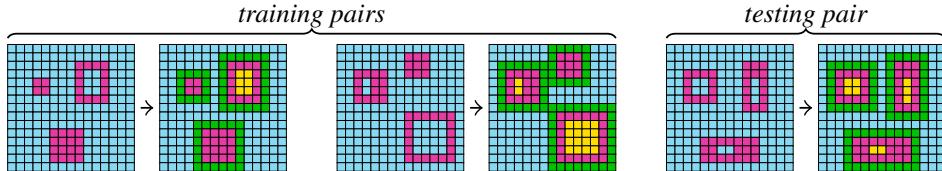


Figure 2: All examples in the original ARC-AGI-1 benchmark suite for puzzle ID 543a7ed5.

2.3 ARC Solvers

Despite the apparent simplicity of the ARC formulation, a surprisingly wide variety of approaches have emerged in an attempt to address it. Many efforts have focused on program-oriented solutions – e.g., *program search* [28, 29], *program synthesis* [30–34], *program sampling* [35], *program repair* [36], and *program induction* [37, 38] – all of which involve the construction of task-specific source codes capable of transforming any subsequent input image. An alternative technique involves the use of variational autoencoders to convert examples into low-dimensional latent vectors, allowing the direct production of output images by applying vector arithmetic [39]. Given that the patterns embodied in most puzzles tend to be simple, the lossless information compression proposed in CompressARC has also demonstrated potential (despite the absence of pretraining) [40], and since certain tasks are more amenable to some techniques than others, a recent implementation considers an ensemble method to exploit their respective puzzle-specific strengths [24]. Finally, both Large Language Models [41–43] and small Transformer models [44, 45] have shown exceptional promise, with newer algorithms employing *test-time training* [46] and the *language of thought hypothesis* [47] to alleviate the shortcomings of these neural architectures.

2.4 ARC-Related Datasets and Generators

A common theme across many ARC solvers is the use of *data augmentation* [48] to produce $\langle \text{input}, \text{output} \rangle$ pairs beyond those originally released by the ARC Prize Foundation. One such effort named ConceptARC [49] introduced a suite of new tasks formed around sixteen specific *concept groups* (e.g., sameness, extraction, counting, etc.). In addition to static problem sets, a handful of (unofficial) procedural generators have been developed to further embellish the sample space. For instance, the BARC implementation relies on synthetic generation using *seeds* (each corresponding to one of the original puzzles) which are then mutated and recombined to produce many thousands of variations. Although the instances produced by these seed generators are intended to reflect those in the official set, only a fraction of tasks are represented, and nearly a third have been shown to produce incorrect results. Finally, the RE-ARC project [23] offers complete task coverage – providing a DSL-based generator for each of the four-hundred training puzzles – but lifts various distributional constraints to increase the diversity of the sample space.³ For illustration, refer to Figure 3 in which several additional degrees of freedom (e.g., the number of boxes, their colors, grid dimensions) have been introduced. Using the nomenclature of Algorithmic Information Theory, one could say that for every original task $\mathcal{T}_i : \mathcal{S}_i \rightarrow \mathcal{S}'_i$, the RE-ARC generator draws its examples from an extended task $\overline{\mathcal{T}}_i$:

$$\overline{\mathcal{T}}_i : \overline{\mathcal{S}}_i \rightarrow \overline{\mathcal{S}}'_i \mid \overline{\mathcal{S}}_i \supseteq \mathcal{S}_i, \overline{\mathcal{T}}_i(s_{i,j}) = \mathcal{T}_i(s_{i,j})$$

Depending on the context, a broader sampling space can offer significant benefits – most notably, allowing the construction and validation of systems that exhibit much stronger generalization capabilities – but it is considerably less useful when evaluating systems explicitly designed to solve the original set of tasks (such as the motivating application we consider in Section §5).

³<https://github.com/michaelhodel/arc-dsl/>

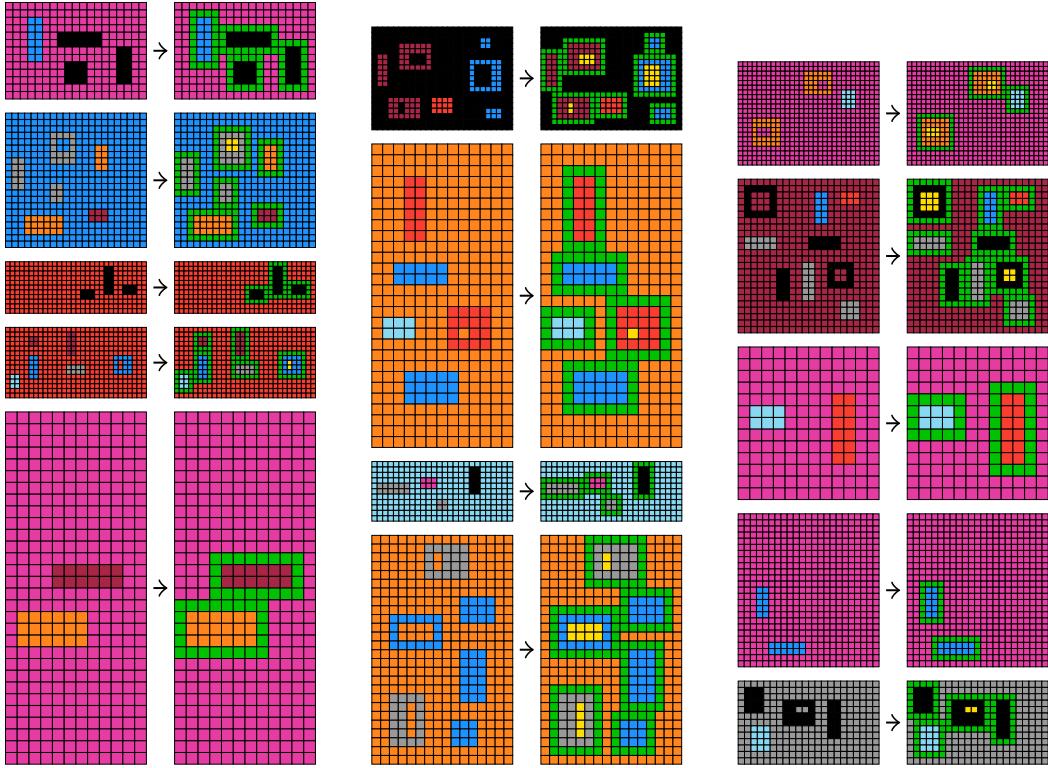


Figure 3: Examples for puzzle ID 543a7ed5 produced by the RE-ARC procedural generator.

3 The ARC-GEN Procedural Benchmark Generator

The focus of our work is the procedural generation of benchmarks that complement the ARC-AGI-1 dataset, subject to two important design criteria. The first is that we seek to construct a generator that is *exhaustive* – that is, covering the complete array of all four-hundred training tasks – which (given the wide range of core knowledge priors expressed in these transformations) is a rather significant undertaking. The second is that we wish for this generator to be *mimetic*, emulating virtually the same constraints and distributional properties that are present in the original suite. The result of these efforts is ARC-GEN, an open-source library that is freely available for both academic and commercial use:

<https://github.com/google/ARC-GEN>

In the following sections, we outline the structure of a typical sub-generator, illustrating the key features and components that differentiate our approach from previous works.

3.1 Parameterization

For each task, our generator defines a parameterized `generate()` function whose arguments dictate the abstract entities contained in an example. We show in Figure 4 the function for puzzle 543a7ed5, which accepts parameters describing the locations, dimensions, and colors of all boxes. These variables are all task-specific; common attributes for other puzzles include pixel coordinates, “sprites” (small bitmap objects), and boolean indicators to toggle certain grid manipulations such as reflection or transposition.

By default, most parameterization arguments are unspecified (using the constant `None`), in which case the generator will automatically populate their values using random numbers. Special care must be taken to ensure that these assignments are consistent with the task – for instance, that boxes do not overlap – and so it is not unusual to find conditional logic in this section dedicated to checking such constraints and (in the case of violation) attempting different combinations of parameter values.

```

def generate(rows=None, cols=None, widths=None, heights=None, colors=None,
            boxes=3, size=15):
    ##### PARAMETERIZATION #####
    if rows is None:
        while True: # rows, cols = box coordinates ; widths, heights = box dimensions
            widths, heights = common.randint(2, 7, boxes), common.randint(2, 7, boxes)
            rows = [common.randint(1, size - height - 1) for height in heights]
            cols = [common.randint(1, size - width - 1) for width in widths]
            if common.overlaps(rows, cols, widths, heights, 2): continue
            colors = [common.pink()] * boxes
            newrows, newcols, newwidths, newheights = [], [], [], []
            for row, col, width, height in zip(rows, cols, widths, heights):
                w, t = common.randint(0, width - 2), common.randint(0, height - 2)
                if not w or not t: continue
                r = row + common.randint(1, height - t - 1)
                c = col + common.randint(1, width - w - 1)
                newrows, newcols = newrows + [r], newcols + [c]
                newwidths, newheights = newwidths + [w], newheights + [t]
            if sum(w * t for w, t in zip(newwidths, newheights)) < 2 * boxes: continue
            rows, cols = rows + newrows, cols + newcols
            widths, heights = widths + newwidths, heights + newheights
            colors.extend([common.yellow()] * len(newrows))
            break

    ##### GENERATION #####
    grid, output = common.grids(size, size, common.cyan())
    for row, col, width, height, color in zip(rows, cols, widths, heights, colors):
        for r in range(row - 1, row + height + 1):
            for c in range(col - 1, col + width + 1):
                if color == common.pink(): output[r][c] = common.green()
                if r < row or r >= row + height or c < col or c >= col + width: continue
                grid[r][c] = color if color == common.pink() else common.cyan()
                output[r][c] = color
    return {"input": grid, "output": output}

    ##### VALIDATION #####
    def validate():
        train = [generate(rows=[2, 4, 10, 3], cols=[8, 3, 5, 9], widths=[4, 2, 4, 2],
                          heights=[5, 2, 4, 3], colors=[6, 6, 6, 4]),
                 generate(rows=[1, 3, 8, 4, 9], cols=[8, 2, 8, 3, 9],
                          widths=[3, 4, 6, 1, 4], heights=[3, 4, 6, 2, 4],
                          colors=[6, 6, 6, 4, 4])]
        test = [generate(rows=[2, 3, 11, 4, 4, 12], cols=[9, 2, 4, 10, 3, 6],
                         widths=[3, 4, 7, 1, 2, 2], heights=[6, 4, 3, 3, 2, 1],
                         colors=[6, 6, 6, 4, 4, 4])]
        return {"train": train, "test": test}

```

Figure 4: Our procedural generation code for just one of the four-hundred puzzles (ID: 543a7ed5).

3.2 Generation

Unlike prior procedural generators for ARC, our core generation logic is decoupled from parameterization, allowing both the *validation* and optional *variation* of tasks (as explained in subsequent sections). The purpose of this generation stanza is to translate all high-level task-specific arguments – whether specified as function parameters or determined synthetically – into pixel-perfect representations of the input and output grids.

In the case of puzzle 543a7ed5, the process is quite simple: we iterate over the set of boxes, and then through all coordinates that each box occupies (marking the appropriate grid cells with the corresponding color); refer to Figure 5 for an array of such examples. For most cases, we have found that the effort required to code the generator for a task is much more straightforward than that required to produce its solution, since we maintain full observability into the output grid and can mask it as necessary in order to create its input in tandem.

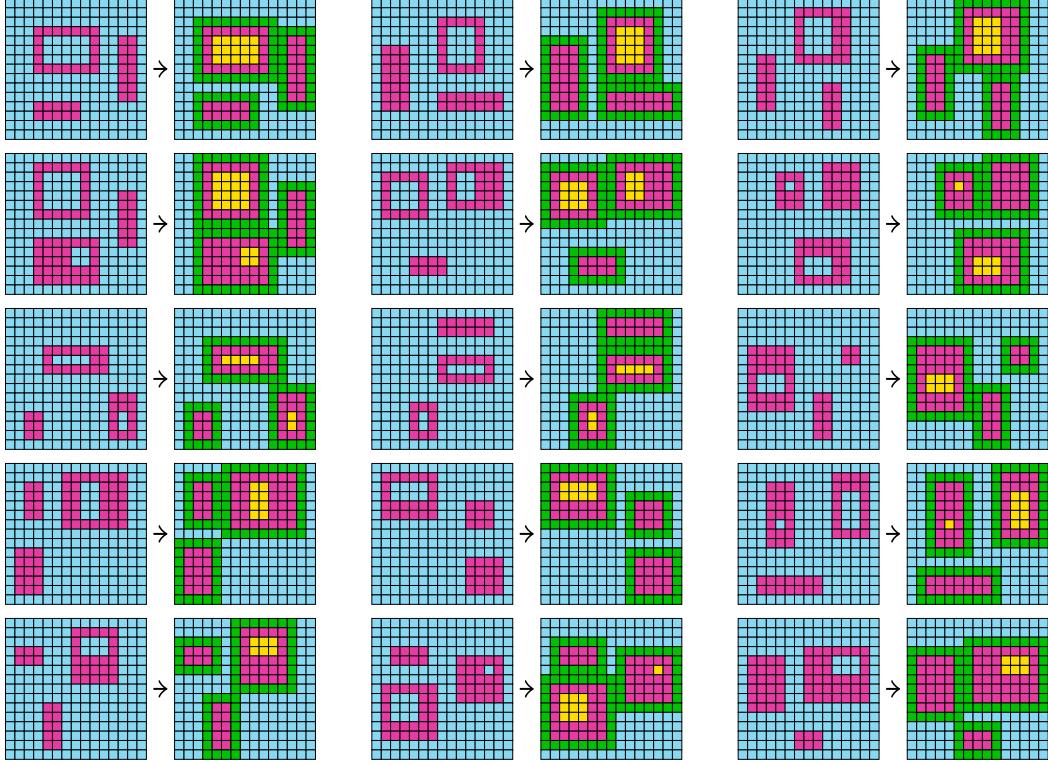


Figure 5: Examples for puzzle ID 543a7ed5 produced by our ARC-GEN procedural generator.

3.3 Validation

The design of any procedural benchmark generator carries the risk of *under-specification*, where the original task \mathcal{T}_i has been inadvertently reduced to one that operates across a narrower range of inputs:

$$\tilde{\mathcal{T}}_i : \tilde{\mathcal{S}}_i \rightarrow \tilde{\mathcal{S}}'_i \mid \tilde{\mathcal{S}}_i \subseteq \mathcal{S}_i, \tilde{\mathcal{T}}_i(\tilde{s}_{i,j}) = \mathcal{T}_i(s_{i,j})$$

Returning to our running example, it would be much easier to implement a generator that strictly produces *closed* boxes, forgoing the additional logic required to properly size and place the occasional holes. However, there is only marginal utility in such a tool, as it might satisfy a simplified transformation that (either accidentally or deliberately) fails when exposed to a wider range of examples drawn from the original sampling space.

In order to demonstrate the behavioral completeness of our library, we include a `validate()` function for each task \mathcal{T}_i that contains the appropriate parameters to reproduce the *entire sequence* of training and test pairs provided in ARC-AGI-1:

$$\mathcal{V}_i(\mathcal{T}_i, \alpha_i) = \mathcal{P}_i = \langle (s_{i,1}, s'_{i,1}), (s_{i,2}, s'_{i,2}), \dots \rangle$$

These validation parameters α_i (all derived manually) also serve as functional unit tests for anyone wishing to submit modifications or improvements to our open-source library, which are always welcome.

3.4 Variations

As noted earlier, there is significant value in employing procedural generators to expand the diversity of examples used to train and evaluate new ARC solvers. In ARC-GEN, we allow such customization for nearly every task, delegating control of the desired variation(s) to callees of our library. We display four such examples in Figure 6 for which the values of `boxes` & `size` have been increased, along with extra overrides to introduce color variations.

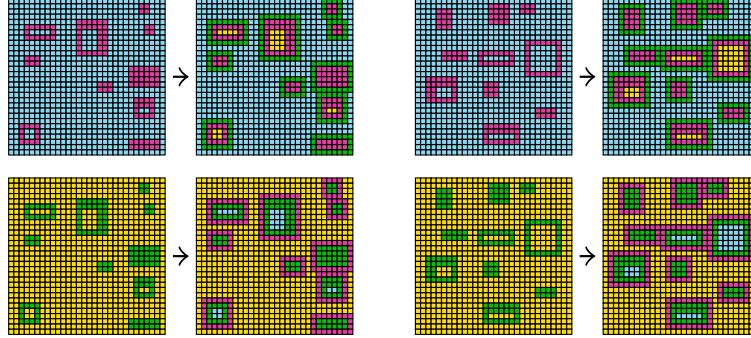


Figure 6: “Large” and “inverted” variations on puzzle ID 543a7ed5 produced by ARC-GEN.

4 Evaluation

In Table 1, we provide a comparison of ARC-GEN vs. the two predominant ARC-flavored generation libraries; specifically, the seeds present in BARC’s synthetic generator [24] and the generation utilities in RE-ARC [23]. For each system, we report its task coverage across the ARC-AGI-1 training puzzles, as well as its validation coverage (which, at present, is provided only by ARC-GEN). Since both BARC and RE-ARC include implementations for the desired transformations (called *source programs* and *verifiers*, respectively), we report the success rate of these programs across the samples produced by each generator. Unsurprisingly, the distributional variance in RE-ARC causes problems for most BARC programs, and the low task representation in BARC (coupled with occasional incompatibility) leads to relatively few successes against the verifiers included with RE-ARC. In contrast, the success rate of ARC-GEN is 100% across all implementations.

5 An Application to the 2025 Google Code Golf Championship

The aforementioned approach of *program synthesis* is a particularly compelling attack vector to solve ARC-AGI, due in part to the many recent advances in automated code generation [50–66]. Yet, aside from DSL-based solutions that defer the majority of grid manipulations to a library of specialized subroutines, there exists no canonical set of reference programs (to our knowledge) that can aid in the training and fine-tuning of ARC-oriented solvers.

To help curate a community-owned collection of pure Pythonic solutions for these tasks, the recent 2025 *Google Code Golf Championship* invited participants from around the world to contribute functional implementations that solve each of the four-hundred ARC-AGI-1 training puzzles.⁴ In order to gamify this process while soliciting the shortest possible implementations, submissions were scored by *program length* – for example, we show in Figure 7 three such source codes (b,e,h) that, while not necessarily minimal, are at least very close to it. Regrettably, our experience administering similar contests suggested that many submissions might have instead resembled the rightmost programs where outputs are hardcoded in various ways, e.g. storing grid values in large integers (c), sparse lists (f), and compressed strings (i). There are an unbounded number of strategies to represent these grids, so it is difficult to disincentivize such submissions if restricting verification to the original ARC-AGI-1 examples.

⁴<https://www.kaggle.com/competitions/google-code-golf-2025>

Table 1: A comparison of procedural benchmark generators for the ARC-AGI-1 dataset.

	Examples from seeds in the BARC Generator [24]	Examples from the RE-ARC Generator [23]	Examples from our ARC-GEN Generator
Task Coverage	27%	100%	100%
Validation Coverage	N/A	N/A	100%
BARC Source Success Rate	100%	3.7%	100%
RE-ARC Verifier Success Rate	16.25%	100%	100%

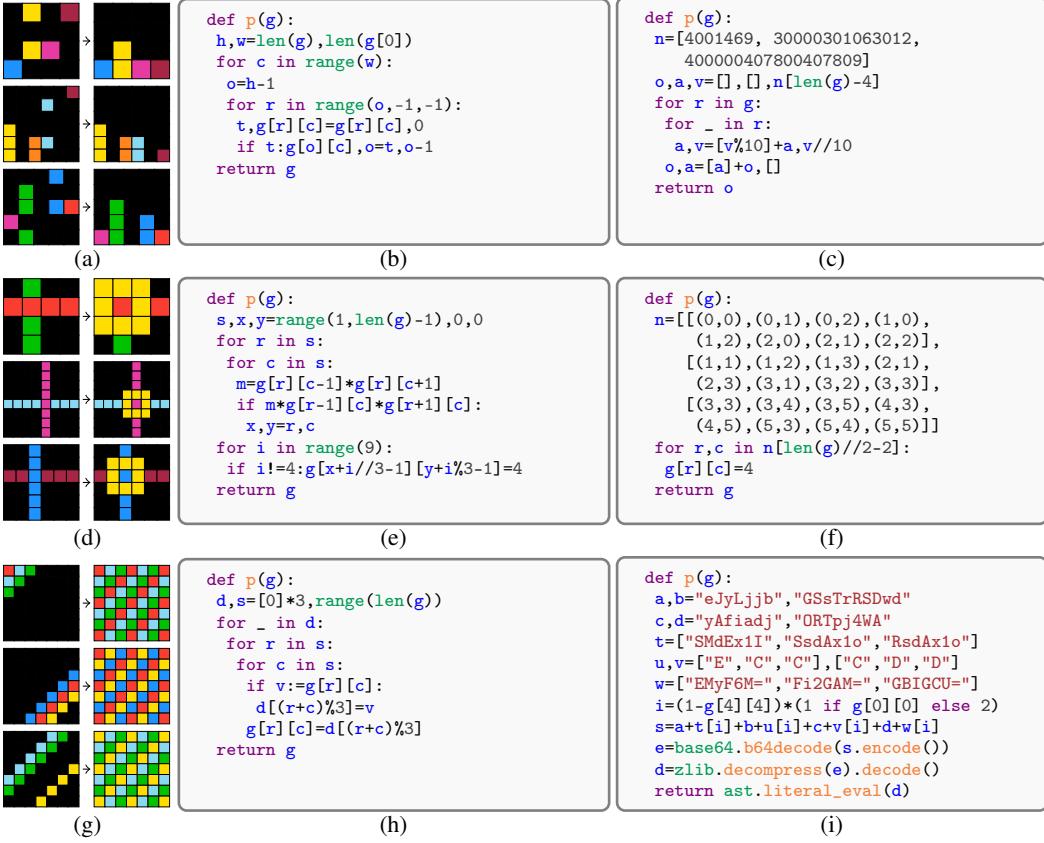


Figure 7: Examples, near-minimal programs, and hardcoded solutions for three ARC tasks.

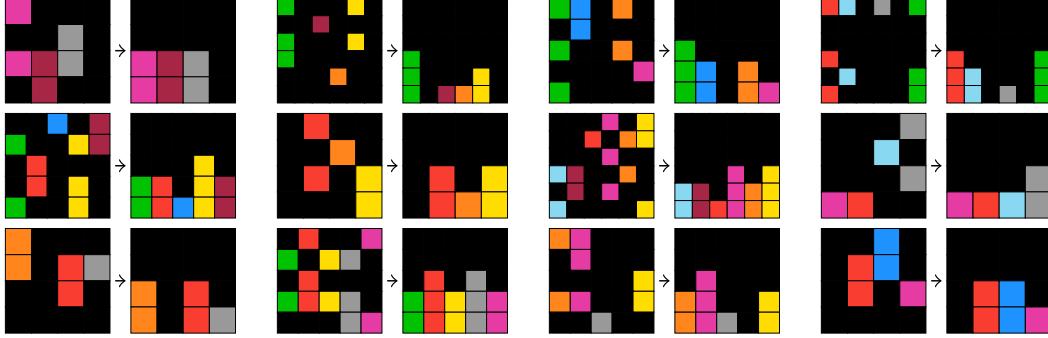
To prevent overfitting, we thus employed ARC-GEN to synthesize *hundreds* of examples per task (totaling 100,000 samples in all), requiring each submission to produce correct outputs across every image pair to be considered for eligibility.⁵ Although it may still be possible to configure hardcoded outputs for individual inputs, such programs are unlikely to be competitive due to their extreme character count.

6 Limitations

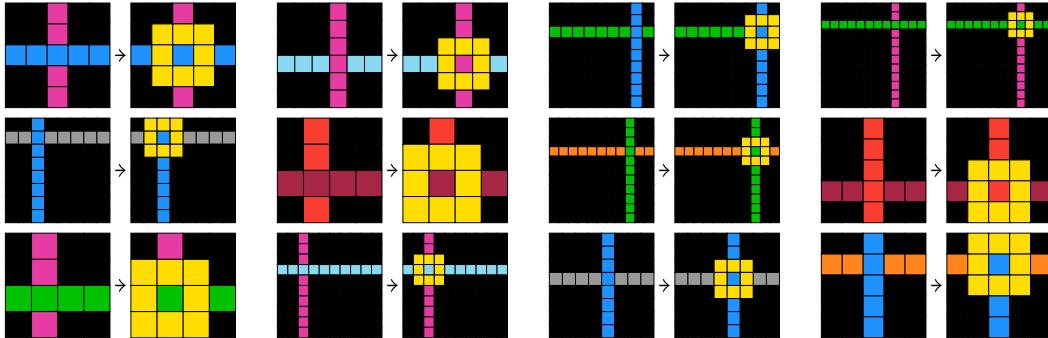
It is important to note that previous efforts target a different design criteria than we do – specifically, amplifying task diversity – which is difficult to capture quantitatively and not reflected in our results. A more apt comparison for that objective might be the efficacy of ARC solvers trained on these samples, which we defer to future studies (as it lies outside the scope of our work). The assessment of these generators is also highly dependent on the availability of functional source programs. Ideally, we would have an broader array of both correct and incorrect programs, which (although currently in short supply) may become more plentiful following the conclusion of our code golf competition. In addition, the limitations of ARC-GEN are tightly intertwined with the spectrum of concepts embodied by ARC-AGI-1. A wider variety of capabilities (i.e., symbolic interpretation, compositional reasoning, and contextual rule application) are emphasized in ARC-AGI-2 [21], and will eventually require the development of an entirely new class of generators. Finally, early previews of ARC-AGI-3 showcase an interactive action-oriented sample space, thus involving variations of *automata synthesis* [67] where generators (albeit significantly more complicated ones) are likely to play a critical role.⁶

⁵<https://www.kaggle.com/datasets/arcgen100k/the-arc-gen-100k-dataset>

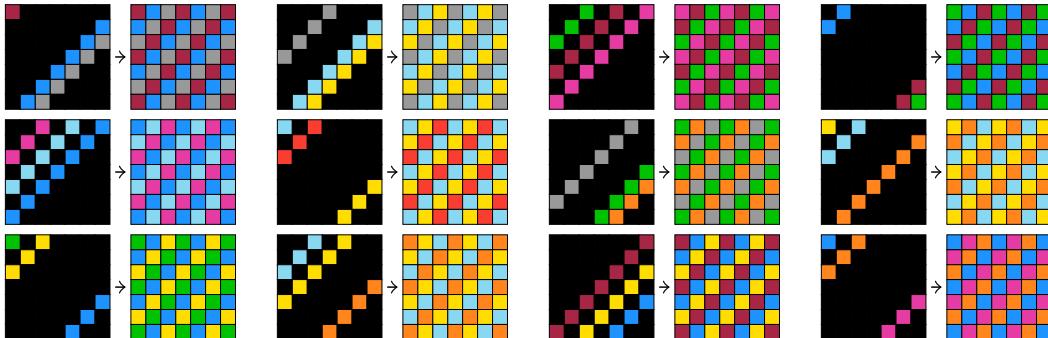
⁶<https://arcprize.org/arc-agi/3/>



(a) A subset of ARC-GEN examples for puzzle ID 1e0a9b12.



(b) A subset of ARC-GEN examples for puzzle ID 67a423a3.



(c) A subset of ARC-GEN examples for puzzle ID 05269061.

Figure 8: Examples for various puzzles produced by our ARC-GEN procedural generator.

7 Conclusion

The ARC-GEN procedural generator presented in this work aims to address the twin goals of *maximizing task coverage* while simultaneously *minimizing distributional deviation* with respect to the seminal ARC-AGI-1 dataset. Each sub-generator included in our open-source release is not only capable of producing new mimetic examples, but also parameterized in such a way that allows the complete reproduction of the original benchmark suite, as well as the creation of a broader variety of tasks. Our library has many potential uses, including (but not limited to) the verification of programs designed to generalize beyond the specific examples included in the original ARC-AGI-1 distribution. We look forward to seeing the use of our tool, along with others, in the pursuit of systems devoted to tackling Artificial General Intelligence.

References

- [1] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolaos Angelopoulos, Tianle Li, Dacheng Li, Banghua Zhu, Hao Zhang, Michael I. Jordan, Joseph E. Gonzalez, and Ion Stoica. Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference. In *Proceedings of the 41st International Conference on Machine Learning*, ICML 2024, pages 8359–8388, 2024. URL <https://proceedings.mlr.press/v235/chiang24b.html>.
- [2] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Tianle Li, Siyuan Zhuang, Zhanghao Wu, Yong-hao Zhuang, Zhuohan Li, Zi Lin, Eric Xing, Joseph E. Gonzalez, Ion Stoica, and Hao Zhang. LMSYS-Chat-1M: A Large-Scale Real-World LLM Conversation Dataset. In *Proceedings of the 12th International Conference on Learning Representations*, ICLR 2024, 2024. URL <https://openreview.net/forum?id=B0fDKxfwt0>.
- [3] Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Benjamin Feuer, Siddhartha Jain, Ravid Schwartz-Ziv, Neel Jain, Khalid Saifullah, Sreemanti Dey, Shubh-Agrawal, Sandeep Singh Sandha, Siddartha Venkat Naidu, Chinmay Hegde, Yann LeCun, Tom Goldstein, Willie Neiswanger, and Micah Goldblum. LiveBench: A Challenging, Contamination-Limited LLM Benchmark. In *Proceedings of the 13th International Conference on Learning Representations*, ICLR 2025, 2025. URL <https://openreview.net/forum?id=sKYHBTaxVa>.
- [4] Priyank Pathak, Shyam Marjit, Shruti Vyas, and Yogesh S Rawat. LR0.FM: Low-Resolution Zero-Shot Classification Benchmark for Foundation Models. In *Proceedings of the 13th International Conference on Learning Representations*, ICLR 2025, 2025. URL <https://openreview.net/forum?id=AsFxRSLtqR>.
- [5] Tsung-Han Wu, Giscard Biamby, Jerome Quenum, Ritwik Gupta, Joseph E. Gonzalez, Trevor Darrell, and David Chan. Visual Haystacks: A Vision-Centric Needle-In-A-Haystack Benchmark. In *Proceedings of the 13th International Conference on Learning Representations*, ICLR 2025, 2025. URL <https://openreview.net/forum?id=9JCNPFL1f9>.
- [6] Chien-yu Huang et al. Dynamic-SUPERB Phase-2: A Collaboratively Expanding Benchmark for Measuring the Capabilities of Spoken Language Models with 180 Tasks. In *Proceedings of the 13th International Conference on Learning Representations*, ICLR 2025, 2025. URL <https://openreview.net/forum?id=s71zZpAW7T>.
- [7] Muhammad A Shah, David Solans Noguero, Mikko A. Heikkilä, Bhiksha Raj, and Nicolas Kourtellis. Speech Robust Bench: A Robustness Benchmark For Speech Recognition. In *Proceedings of the 13th International Conference on Learning Representations*, ICLR 2025, 2025. URL <https://openreview.net/forum?id=DOLuQNZfE1>.
- [8] Guo Chen, Yicheng Liu, Yifei Huang, Baoqi Pei, Jilan Xu, Yuping He, Tong Lu, Yali Wang, and Limin Wang. CG-Bench: Clue-grounded Question Answering Benchmark for Long Video Understanding. In *Proceedings of the 13th International Conference on Learning Representations*, ICLR 2025, 2025. URL <https://openreview.net/forum?id=le4IoZZHy1>.
- [9] Fei Ye, Zaixiang Zheng, Dongyu Xue, Yuning Shen, Lihao Wang, Yiming Ma, Yan Wang, Xinyou Wang, Xiangxin Zhou, and Quanquan Gu. ProteinBench: A Holistic Evaluation of Protein Foundation Models. In *Proceedings of the 13th International Conference on Learning Representations*, ICLR 2025, 2025. URL <https://openreview.net/forum?id=BksqWM8737>.
- [10] Dhruv Gautam, Spandan Garg, Jinu Jang, Neel Sundaresan, and Roshanak Zilouchian Moghadam. RefactorBench: Evaluating Stateful Reasoning in Language Agents Through Code. In *Proceedings of the 13th International Conference on Learning Representations*, ICLR 2025, 2025. URL <https://openreview.net/forum?id=NiNIthntx7>.
- [11] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code. In *Proceedings of the 13th International Conference on Learning Representations*, ICLR 2025, 2025. URL <https://openreview.net/forum?id=chfJJYC3iL>.

- [12] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? In *Proceedings of the 12th International Conference on Learning Representations*, ICLR 2024, 2024. URL <https://openreview.net/forum?id=VTF8yNQM66>.
- [13] Elliot Glazer, Ege Erdil, Tamay Besiroglu, et al. FrontierMath: A Benchmark for Evaluating Advanced Mathematical Reasoning in AI, 2024. URL <https://arxiv.org/abs/2411.04872>.
- [14] Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, et al. Humanity’s Last Exam, 2025. URL <https://arxiv.org/abs/2501.14249>.
- [15] François Chollet. On the Measure of Intelligence, 2019. URL <https://arxiv.org/abs/1911.01547>.
- [16] Addison Howard, Eunbyung Park, and Wendy Kan. ImageNet Object Localization Challenge. <https://kaggle.com/competitions/imagenet-object-localization-challenge>, 2018. Kaggle.
- [17] Solim LeGris, Wai Keen Vong, Brenden M. Lake, and Todd M. Gureckis. H-ARC: A Robust Estimate of Human Performance on the Abstraction and Reasoning Corpus Benchmark, 2024. URL <https://arxiv.org/abs/2409.01374>.
- [18] Gustaw Opiełka, Hannes Rosenbusch, Veerle Vijverberg, and Claire E. Stevenson. Do Large Language Models Solve ARC Visual Analogies Like People Do?, 2024. URL <https://arxiv.org/abs/2403.09734>.
- [19] François Chollet, Mike Knoop, Bryan Landers, Greg Kamradt, Hansueli Jud, Walter Reade, and Addison Howard. ARC Prize 2024. <https://kaggle.com/competitions/arc-prize-2024>, 2024. Kaggle.
- [20] François Chollet, Mike Knoop, Gregory Kamradt, and Bryan Landers. ARC Prize 2024: Technical Report, 2024. URL <https://arxiv.org/abs/2412.04604>.
- [21] Francois Chollet, Mike Knoop, Gregory Kamradt, Bryan Landers, and Henry Pinkard. ARC-AGI-2: A New Challenge for Frontier AI Reasoning Systems, 2025. URL <https://arxiv.org/abs/2505.11831>.
- [22] François Chollet, Mike Knoop, Greg Kamradt, Walter Reade, and Addison Howard. ARC Prize 2025. <https://kaggle.com/competitions/arc-prize-2025>, 2025. Kaggle.
- [23] Michael Hodel. Addressing the Abstraction and Reasoning Corpus via Procedural Example Generation, 2024. URL <https://arxiv.org/abs/2404.07353>.
- [24] Wen-Ding Li, Keya Hu, Carter Larsen, Yuqing Wu, Simon Alford, Caleb Woo, Spencer M. Dunn, Hao Tang, Wei-Long Zheng, Yewen Pu, and Kevin Ellis. Combining Induction and Transduction for Abstract Reasoning. In *Proceedings of the 13th International Conference on Learning Representations*, ICLR 2025, 2025. URL <https://openreview.net/forum?id=UmdotAAVDe>.
- [25] Samuel Acquaviva, Yewen Pu, Marta Kryven, Theodoros Sechopoulos, Catherine Wong, Gabrielle E Ecanow, Maxwell Nye, Michael Henry Tessler, and Joshua B. Tenenbaum. Communicating Natural Programs to Humans and Machines. In *Proceedings of the 36th Conference on Neural Information Processing Systems*, NeurIPS 2022, 2022. URL <https://dl.acm.org/doi/10.5555/3600270.3600540>.
- [26] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. In *Proceedings of the 34th Conference on Neural Information Processing Systems*, NeurIPS 2020, pages 1877–1901, 2020. URL <https://dl.acm.org/doi/abs/10.5555/3495724.3495883>.

- [27] Sachin Ravi and Hugo Larochelle. Optimization as a Model for Few-Shot Learning. In *Proceedings of the 5th International Conference on Learning Representations*, ICLR 2017, 2017. URL <https://openreview.net/forum?id=rJY0-Kc11>.
- [28] Simon Alford, Anshula Gandhi, Akshay Rangamani, Andrzej Banburski, Tony Wang, Sylee Dandekar, John Chin, Tomaso Poggio, and Peter Chin. Neural-Guided, Bidirectional Program Search for Abstraction and Reasoning. In *Proceedings of the 10th International Conference on Complex Networks and Their Applications*, pages 657–668, 2022.
- [29] Matthew Macfarlane and Clément Bonnet. Searching Latent Program Spaces. In *ICML 2025 Workshop on Programmatic Representations for Agent Learning*, 2025. URL <https://openreview.net/forum?id=Va937YZ9it>.
- [30] James Ainooson, Deepayan Sanyal, Joel P. Michelson, Yuan Yang, and Maithilee Kunda. A Neurodiversity-Inspired Solver for the Abstraction & Reasoning Corpus (ARC) Using Visual Imagery and Program Synthesis, 2023. URL <https://arxiv.org/abs/2302.09425>.
- [31] Mikel Bober-Irizar and Soumya Banerjee. Neural networks for abstraction and reasoning. *Scientific Reports*, 14, 2024. URL <https://doi.org/10.1038/s41598-024-73582-7>.
- [32] Céline Hocquette and Andrew Cropper. Relational Decomposition for Program Synthesis, 2025. URL <https://arxiv.org/abs/2408.12212>.
- [33] Di Huang, Ziyuan Nan, Xing Hu, Pengwei Jin, Shaohui Peng, Yuanbo Wen, Rui Zhang, Zidong Du, Qi Guo, Yewen Pu, and Yunji Chen. ANPL: Towards Natural Programming with Interactive Decomposition. In *Proceedings of the 37th Conference on Neural Information Processing Systems*, NeurIPS 2023, 2023. URL <https://openreview.net/forum?id=RTRs3ZTsSj>.
- [34] Jonas Witt, Sebastijan Dumančić, Tias Guns, and Claus-Christian Carbon. A Divide, Align, and Conquer Strategy for Program Synthesis. *Journal of Artificial Intelligence Research*, 82: 1961–1997, March 2025. URL <https://doi.org/10.1613/jair.1.16847>.
- [35] Natasha Butt, Blazej Manczak, Auke Wiggers, Corrado Rainone, David W. Zhang, Michaël Deferrard, and Taco Cohen. CodeIt: Self-Improving Language Models with Prioritized Hindsight Replay. In *Proceedings of the 41st International Conference on Machine Learning*, ICML 2024, pages 5013–5034, 2024. URL <https://proceedings.mlr.press/v235/butt24a.html>.
- [36] Hao Tang, Keya Hu, Jin Peng Zhou, Si Cheng Zhong, Wei-Long Zheng, Xujie Si, and Kevin Ellis. Code Repair with LLMs gives an Exploration-Exploitation Tradeoff. In *Proceedings of the 38th Conference on Neural Information Processing Systems*, NeurIPS 2024, 2024. URL <https://openreview.net/forum?id=o863gX6DxA>.
- [37] Simon Ouellette. Towards Efficient Neurally-Guided Program Induction for ARC-AGI, 2024. URL <https://arxiv.org/abs/2411.17708>.
- [38] Ruocheng Wang, Eric Zelikman, Gabriel Poesia, Yewen Pu, Nick Haber, and Noah Goodman. Hypothesis Search: Inductive Reasoning with Language Models. In *Proceedings of the 12th International Conference on Learning Representations*, ICLR 2024, 2024. URL <https://openreview.net/forum?id=G7UtIGQmjm>.
- [39] Karel Veldkamp, Hannes Rosenbusch, Luca Thoms, and Claire Stevenson. Solving ARC visual analogies with neural embeddings and vector arithmetic: A generalized method. In *OSF*, 2023. URL <https://doi.org/10.17605/OSF.IO/AKP86>.
- [40] Isaac Liao and Albert Gu. ARC-AGI Without Pretraining, 2025. URL https://iliao2345.github.io/blog_posts/arc_agi_without_pretraining/arc_agi_without_pretraining.html.
- [41] Jeremy Berman. How I came in first on ARC-AGI-Pub using Sonnet 3.5 with Evolutionary Test-time Compute, 2024. URL <https://jeremyberman.substack.com/p/how-i-got-a-record-536-on-arc-agi>.
- [42] Ryan Greenblatt. Getting 50% (SoTA) on ARC-AGI with GPT-4o, 2024. URL <https://redwoodresearch.substack.com/p/getting-50-sota-on-arc-agi-with-gpt>.

- [43] John Chong Min Tan and Mehul Motani. Large Language Model (LLM) as a System of Multiple Expert Agents: An Approach to solve the Abstraction and Reasoning Corpus (ARC) Challenge, 2023. URL <https://arxiv.org/abs/2310.05146>.
- [44] Paul Fletcher-Hill. Mini-ARC: Solving Abstraction and Reasoning Puzzles with Small Transformer Models, 2024. URL <https://www.paulfletcherhill.com/mini-arc.pdf>.
- [45] Jean-Francois Puget. A 2D nGPT Model For Arc Prize, 2024. URL <https://github.com/jfpuget/ARC-AGI-Challenge-2024/blob/main/arc.pdf>.
- [46] Ekin Akyürek, Mehul Damani, Adam Zweiger, Linlu Qiu, Han Guo, Jyothish Pari, Yoon Kim, and Jacob Andreas. The Surprising Effectiveness of Test-Time Training for Few-Shot Learning, 2025. URL <https://arxiv.org/abs/2411.07279>.
- [47] Seungpil Lee, Woochang Sim, Donghyeon Shin, Wongyu Seo, Jiwon Park, Seokki Lee, Sanha Hwang, Sejin Kim, and Sundong Kim. Reasoning Abilities of Large Language Models: In-Depth Analysis on the Abstraction and Reasoning Corpus. *ACM Transactions on Intelligent Systems and Technology*, January 2025. URL <https://doi.org/10.1145/3712701>.
- [48] Daniel Franzen, Jan Disselhoff, and David Hartmann. The LLM ARCHitect: Solving ARC-AGI Is A Matter of Perspective, 2024. URL https://github.com/da-fr/arc-prize-2024/blob/main/the_architects.pdf.
- [49] Arsenii Kirillovich Moskvichev, Victor Vikram Odonard, and Melanie Mitchell. The ConceptARC Benchmark: Evaluating Understanding and Generalization in the ARC Domain. *Transactions on Machine Learning Research*, 2023. URL <https://openreview.net/forum?id=8ykyGbtt2q>.
- [50] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program Synthesis with Large Language Models, 2021. URL <https://arxiv.org/abs/2108.07732>.
- [51] Matthew Bowers, Theo X. Olausson, Lionel Wong, Gabriel Grand, Joshua B. Tenenbaum, Kevin Ellis, and Armando Solar-Lezama. Top-Down Synthesis for Library Learning. In *Proceedings of the 50th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2023, pages 1182–1213, 2023. URL <https://doi.org/10.1145/3571234>.
- [52] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, et al. Evaluating Large Language Models Trained on Code, 2021. URL <https://arxiv.org/abs/2107.03374>.
- [53] Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdel-rahman Mohamed, and Pushmeet Kohli. RobustFill: Neural Program Learning under Noisy I/O. In *Proceedings of the 34th International Conference on Machine Learning*, ICML 2017, pages 990–998, 2017. URL <https://proceedings.mlr.press/v70/devlin17a.html>.
- [54] Xueying Du, Mingwei Liu, Kaixin Wang, Hanlin Wang, Junwei Liu, Yixuan Chen, Jiayi Feng, Chaofeng Sha, Xin Peng, and Yiling Lou. Evaluating Large Language Models in Class-Level Code Generation. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE 2024, 2024. URL <https://doi.org/10.1145/3597503.3639219>.
- [55] Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Lucas Morales, Luke Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B. Tenenbaum. 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*, PLDI 2021, pages 835–850, 2021. URL <https://doi.org/10.1145/3453483.3454080>.
- [56] Richard Evans, Matko Bošnjak, Lars Buesing, Kevin Ellis, David Pfau, Pushmeet Kohli, and Marek Sergot. Making sense of raw input. *Artificial Intelligence*, 299:103521, 2021. URL <https://doi.org/10.1016/j.artint.2021.103521>.

- [57] Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Scott Yih, Luke Zettlemoyer, and Mike Lewis. InCoder: A Generative Model for Code Infilling and Synthesis. In *Proceedings of the 11th International Conference on Learning Representations*, ICLR 2023, 2023. URL <https://openreview.net/forum?id=hQwb-1bM6EL>.
- [58] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided Language Models. In *Proceedings of the 40th International Conference on Machine Learning*, ICML 2023, pages 10764–10799, 2023. URL <https://proceedings.mlr.press/v202/gao23f.html>.
- [59] Chengshu Li, Jacky Liang, Andy Zeng, Xinyun Chen, Karol Hausman, Dorsa Sadigh, Sergey Levine, Li Fei-Fei, Fei Xia, and Brian Ichter. Chain of Code: Reasoning with a Language Model-Augmented Code Emulator. In *Proceedings of the 41st International Conference on Machine Learning*, ICML 2024, pages 28259–28277, 2024. URL <https://proceedings.mlr.press/v235/li24ar.html>.
- [60] Wen-Ding Li and Kevin Ellis. Is Programming by Example Solved by LLMs? In *Proceedings of the 38th Conference on Neural Information Processing Systems*, NeurIPS 2024, 2024. URL <https://openreview.net/forum?id=xqc8yyhScL>.
- [61] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittweiser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d’Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-Level Code Generation with AlphaCode. *Science*, 378(6624):1092–1097, 2022. URL <https://www.science.org/doi/abs/10.1126/science.abq1158>.
- [62] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis. In *Proceedings of the 11th International Conference on Learning Representations*, ICLR 2023, 2023. URL https://openreview.net/forum?id=iaYcJKpY2B_.
- [63] Ruizhong Qiu, Weiliang Will Zeng, James Ezick, Christopher Lott, and Hanghang Tong. How efficient is LLM-generated code? A rigorous & high-standard benchmark. In *Proceedings of the 13th International Conference on Learning Representations*, ICLR 2025, 2025. URL <https://openreview.net/forum?id=suz4utPr9Y>.
- [64] Jürgen Schmidhuber. Optimal Ordered Problem Solver. *Machine Learning*, 54(3):211–254, March 2004. URL <https://doi.org/10.1023/B:MACH.0000015880.99707.b2>.
- [65] Hao Tang and Kevin Ellis. From Perception to Programs: Regularize, Overparameterize, and Amortize. In *Proceedings of the 40th International Conference on Machine Learning*, ICML 2023, pages 33616–33631, 2023. URL <https://proceedings.mlr.press/v202/tang23c.html>.
- [66] Hao Tang, Darren Yan Key, and Kevin Ellis. WorldCoder, a Model-Based LLM Agent: Building World Models by Writing Code and Interacting with the Environment. In *Proceedings of the 38th Conference on Neural Information Processing Systems*, NeurIPS 2024, 2024. URL <https://openreview.net/forum?id=QGJSXMhVaL>.
- [67] Ria Das, Joshua B. Tenenbaum, Armando Solar-Lezama, and Zenna Tavares. Combining Functional and Automata Synthesis to Discover Causal Reactive Programs. In *Proceedings of the 50th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2023, pages 1628–1658, 2023. URL <https://doi.org/10.1145/3571249>.

A Appendix: Reproducing the {Generator x Program} Success Rates

To reproduce the success rates in Table 1, first download our ARC-GEN-100K dataset and clone various GitHub repositories as follows:

```
$ curl -L -o ARC-GEN-100K.zip \
  https://www.kaggle.com/api/v1/datasets/download/arcgen100k/the-arc-gen-100k-dataset
$ git clone --recurse-submodules https://github.com/google/ARC-GEN.git
$ git clone https://github.com/michaelhodel/re-arc.git
$ git clone https://github.com/xu3kev/BARC.git
```

Then, extract the evaluation utilities and unzip all example benchmarks:

```
$ cp -r ARC-GEN/misc/evaluation/* . && mkdir examples && cd examples
$ mkdir ARC-GEN-100K && cd ARC-GEN-100K && unzip ../../ARC-GEN-100K.zip && cd ..
$ unzip ../BARC.zip && unzip ../re-arc/re_arc.zip && cd ..
```

Evaluation using the RE-ARC verifiers should be done from the `re-arc` directory, e.g.:

```
$ cd re-arc

$ python3 evaluate_using_re_arc.py ../examples/BARC/tasks
Testing task 007bbfb7 ... pass
Testing task 00d62c1b ... pass
Testing task 017c7c7b ... FAIL
[...]
Examples pass for 65/400 tasks (16.25%)

$ python3 evaluate_using_re_arc.py ../examples/ARC-GEN-100K
Testing task 007bbfb7 ... pass
Testing task 00d62c1b ... pass
[...]
Examples pass for 400/400 tasks (100%)

$ cd ..
```

Evaluation using the BARC source programs should be done from the `BARC` directory, e.g.:

```
$ cd BARC

$ python3 -O evaluate_using_barc.py --exemplifiedir=../examples/re_arc/tasks
Testing task 007bbfb7 ... pass
Testing task 00d62c1b ... FAIL
[...]
Examples pass for 4/108 tasks (3.7%)

$ python3 -O evaluate_using_barc.py --exemplifiedir=../examples/ARC-GEN-100K
Testing task 007bbfb7 ... pass
Testing task 00d62c1b ... pass
[...]
Examples pass for 108/108 tasks (100%)

$ cd ..
```

We recommend performing these experiments in a standard Linux environment, but there are no specific hardware requirements (e.g., CPU speed, number of cores, RAM, etc). Please allocate up to six hours of runtime for all verifiers to complete.