

---

# DON'T THROW THE BABY OUT WITH THE BATHWATER: HOW AND WHY DEEP LEARNING FOR ARC

---

Jack Cole<sup>1</sup>, Mohamed Osman<sup>1</sup>  
MindsAI<sup>2</sup>  
Tufa Labs  
{jack, mohamed}@tufalabs.ai

## ABSTRACT

The Abstraction and Reasoning Corpus (ARC-AGI) presents a formidable challenge for AI systems. Despite the typically low performance on ARC, the deep learning paradigm remains the most effective known strategy for generating skillful (state-of-the-art) neural networks (NN) across varied modalities and tasks in vision, language etc. The deep learning paradigm has proven to be able to train these skillful neural networks and learn the abstractions needed in these diverse domains. Our work doubles down on that and continues to leverage this paradigm by incorporating on-the-fly NN training at test time.

We demonstrate that fully committing to deep learning's capacity to acquire novel abstractions yields state-of-the-art performance on ARC. Specifically, we treat both the neural network and the optimizer (rather than just a pre-trained network) as integral components of the inference process, fostering generalization to unseen tasks.

Concretely, we propose a methodology for training on ARC, starting from pretrained LLMs, and enhancing their ARC reasoning. We also propose Test-Time Fine-Tuning (TTFT) and the Augment Inference Reverse-Augmentation and Vote (AIRV) as effective test-time techniques. We are the first to propose and show deep learning can be used effectively for ARC, showing boosts of up to 260% in accuracy with AIRV and a further 300% boost with TTFT. An early version of this approach secured first place in the 2023 ARCathon competition, while the final version achieved the current best score on the ARC private test-set (58%).

Our findings highlight the key ingredients of a robust reasoning system in unfamiliar domains, underscoring the central mechanisms that improve broad perceptual reasoning.

## 1 Introduction

Some criticisms of the current deep learning (DL) paradigm rightly note that current best models and methods are overfit to the popular datasets [1]. The limitations of testing on large datasets have become apparent. For example, performance on ImageNet has exceeded saturation, where models are progressing by learning patterns in the biases that labelers of ImageNet tended to make [2]. This is an indication of a gap in the current testing paradigm. Trained models are tested on data that is similar to the data they were trained on (in-distribution data). Models are evaluated on existing skills and we neglect benchmarking the efficiency of the learning process itself [1].

Alternatively, consider the example of this single riddle from the Abstraction and Reasoning Corpus (ARC) [1], shown in Figure 1. Any system attempting to solve the riddle must work from the three provided examples and should infer that it is necessary to transform each colored square into its corresponding pattern. Due to the simplicity of this dataset's setup, little pre-training knowledge can be leveraged to solve these riddles. Instead, the solver must "figure out" the transformation based on the few examples provided. Crucially, the model must learn a new transformation, which limits

---

<sup>1</sup>The MindsAI team in the ARC-AGI 2024 competition also consisted of Michael Hodel who has had a significant impact on the team's success in 2024.

<sup>2</sup>MindsAI constitutes the core research team at Tufa Labs (<https://www.tufalabs.ai/>)

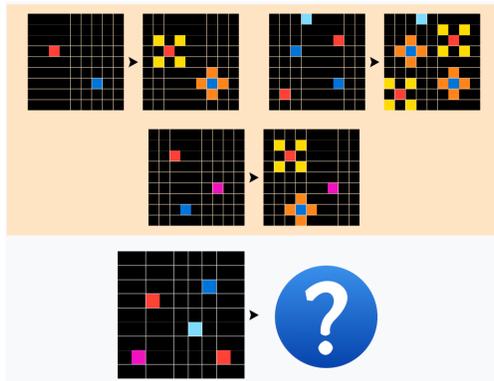


Figure 1: An example of a single easy ARC task (one datapoint). This task is solved by surrounding the red pixels with four yellow corners, blue pixels with four orange side pixels, whereas cyan and magenta input pixels remain unchanged.

the degree to which the model can rely on pretrained knowledge or zero shot performance. This puts a heavy emphasis on the need for contextual reasoning (reaching the correct associations) during the evaluation. Therefore, this dataset becomes a reliable test of the efficiency of the learning process itself.

Large-scale foundation models (both vision based and LLMs) do not perform well out of the box, when prompted with these problems [3, 4]. Moreover, even though neural nets generally achieve state-of-the-art in natural language processing and difficult visual classification and detection tasks [5, 6], these are perceptual/qualitative-type problems that require highly contextual and dynamic reasoning. There is significant uncertainty in the research community on whether neural nets can ever be trained or enhanced to perform well on ARC or similar datasets [1, 3].

Our perspective that these ARC problems are actually more “perceptual” and qualitative than quantitative in nature. There are no performant quantitative approaches to searching the space of possible input to output transformations since there are near infinite possible transformations even with only a few basic priors [1].

Learning performant abstractions from data is what the deep learning paradigm is known for, specifically when applied on difficult perceptual or qualitative problems. Deep learning solutions produce state-of-the-art results on perceptual problems, for example NLP and vision. The deep learning paradigm consists of an untrained neural network combined with an optimizer (e.g., AdamW, SGD). When these two components are enabled with sufficient amounts of data and compute, highly-skilled (accurate), artifacts are produced. Artifacts that possess the right abstractions for the task at hand. The artifacts we refer to are the trained neural networks.

This points us to the idea that this paradigm, namely both the untrained NN **and** the optimizer algorithm (as opposed to just a well-trained NN) can be what creates the novel abstractions needed for correct predictions on the ARC private test set. Indeed, we are the first to find success on ARC by implementing this combination of optimizer and NN in the evaluation loop.

We are able to explore what kind of training data, architecture decisions, model size, and other factors that impact test time tuning and the model’s ability to create abstractions on the fly, to solve novel ARC tasks in the forward pass.

We contribute the following:

- We motivate and present architecture and pre-training recipe decisions for a performant ARC neural network in subsection 3.1.
- We propose the methodology for creating training data for Test-Time Fine-Tuning (TTFT) in subsection 3.2.
- We motivate and propose the Augment Inference Reverse-augmentation and Vote (AIRV) and (TTFT) as test-time methods for improving ARC performance (section 3), showing a 2.6 fold increase and a further 3 fold increase over baseline ARC-pre-training in the ARC private set accuracy respectively.
- This method helps achieve first place in the 2023 ARCathon competition, and achieves the highest score on the ARC private test set during the 2024 ARC kaggle competition. Unlike previous work, we achieve this on the completely novel ARC problems in the ARC private test set. We achieve the best score in the time and compute restricted kaggle test environment [7].

## 2 Problem and desired properties of a solver

### 2.1 Dataset description

The *Abstraction and Reasoning Corpus* (ARC) dataset  $\mathcal{D}$  consists of a collection of tasks (also called riddles in this paper)  $T_{i=1}^N$ , where each task  $T_i$  is defined as follows:

- A set of training examples  $(x_j^{(i)}, y_j^{(i)})_{j=1}^{n_i}$ , where each  $x_j^{(i)}$  and  $y_j^{(i)}$  are input and output grids, for each task  $T_i$  respectively. The input and output grids together are referred to as a grid-pair or example. Both terms are used interchangeably.
- A set of test inputs  $x_k^{(i)}_{k=1}^{m_i}$ , with corresponding outputs  $y_k^{(i)}_{k=1}^{m_i}$  to be predicted.

Each grid is a 2D array  $x \in C^{h \times w}$  of variable height  $h$  and width  $w$ , where  $C$  is a set of 10 colors. The number of training examples  $n_i$  and test examples  $m_i$  are variable across tasks and usually range from 2-6.

The objective is to infer a task-specific function  $f_i$  such that  $f_i(x_j^{(i)}) = y_j^{(i)}$  for all training examples, and then apply  $f_i$  to the test inputs to obtain the test outputs.

### 2.2 What does a solver need to excel on ARC?

Each riddle is akin to a small dataset of input-to-output examples. A solver must use associative learning as a part of the process of solving the riddle. Solvers must develop associations between individual input and output grids and across the different input and output grid pairs, then apply the relevant learned associations to the given test input grid.

In contrast to this, both vision-based meta-learning and natural language based reasoning datasets have a memorization problem. High performing methods on those datasets were found to learn generic features that allow very high levels of accuracy without significant meta-learning. These methods were found to rely more on pretraining knowledge rather than meta-learning to gain new skills on the new tasks provided by these datasets. [8, 9]

This is possible in those datasets because the tasks share a lot of common structure, for example the Mini-Imagenet dataset [10], where the subtasks are all image-classification-based and good general object representations can be learned and reused. This zero-shot feature reuse was shown to take place with the MAML algorithm on Mini-ImageNet [8]. These zero-shot models can outperform meta-learning models on these tasks, holding state-of-the-art results [11, 9].

For a good ARC solver, the opposite of this is desirable. Encoding supposedly good features can lead to incorrect assumptions and missing relevant details. Instead, it is desirable to encode a more sophisticated learning process that can reason about the new examples and the possible transformations. In-context-learning (ICL) is an initial candidate here [12, 13]. The ARC dataset is a challenging test of whether a system can perform this more true form of learning from a few examples.

A certain amount of flexibility and precision will be necessary within the inner workings of a solver, to satisfy the above requirements and perform with high accuracy. The solver needs to identify how relevant points in the input get transformed, including all the rules involved. The solver then needs to be dynamic enough to not only develop representations, but also access those dynamically created representations to be able to correctly apply them in the new context (test input pair).

### 2.3 The associative learning ability in the forward pass of a solver

Not all few-shot learning or meta-learning algorithms are well-suited for ARC. Some architectures, such as zero-shot learners with shallow mixing, primarily rely on generating independent deep embeddings for individual grids. These embeddings are then combined in a relatively simple manner to produce an output grid or a classification result. One example of this is [14].

One example of such an ill-suited architecture is Proto-Net [15]. In a straightforward application of Proto-Net to ARC, each input-output grid-pair is embedded separately into a vector, and these vectors are averaged to create a "prototypical" representation. While this approach allows for a simple projection-based inference, it lacks the ability to capture essential interactions across grid-pairs

To effectively reason about transformations across multiple examples, a model needs to process all grid-pairs in unison rather than simply averaging individual representations. Without this capability, the model may overlook crucial shared

patterns and transformations necessary for solving the task. The limitations of such shallow interaction architectures become particularly evident in tasks where multiple examples must be considered together to infer a general rule.

In the proto-net example model, the only cross grid-pair interaction is the averaging of the embeddings. This averaging is not complex enough and even may destroy information inadvertently, due to the diversity of riddle objectives possible.

Consider a riddle that mirrors red objects horizontally and blue objects vertically. If the test input contains both red and blue objects, a solver must see and recognize these transformations jointly. Shallow architectures like CodeIt [16] struggle with such tasks because they lack mechanisms for simultaneous reasoning across all grid-pairs.

However, incorporating a structured form of associative learning in the forward pass enables a solver to process grid-pairs holistically. This is further discussed in Section 3.1, where we explore how structured cross-grid interactions can significantly enhance generalization and performance. Another example of this, applied step-by-step to an ARC riddle, is worked through in more detail in 3.2

### 3 Solution

#### 3.1 Solution Part 1: emphasizing In-Context-Learning (ICL)

##### 3.1.1 Associative learning in LLMs' forward pass

Recent work has uncovered growing evidence that large language models (LLMs) engage in a form of associative learning [13, 17]. For instance, there is now substantial support for the idea that LLMs can identify and utilize Probabilistic Context-Free Grammars (PCFGs), effectively operating as versatile pattern-recognition tools [4]. In natural language, contextual nuances play a crucial role—each word's meaning can shift dramatically based on its placement within a sentence—so extensive in-context learning appears necessary for these models to generate tokens that are both coherent and context-appropriate. Collectively, these observations indicate that LLMs can establish and exploit relationships among input tokens, an essential ingredient for achieving strong performance on ARC tasks.

[18] find that models like BART and T5 can represent and track changing entity states over the course of a narrative (e.g., whether an object is empty or who possesses it), even without explicit supervision. Their analysis also shows that these representations tend to be localized in specific tokens and can be directly manipulated, causing the model to update its subsequent text generation accordingly. Crucially, most of this dynamic-tracking ability comes from extensive open-domain pre-training rather than fine-tuning, indicating that LLMs may possess the requisite capacity motivated in subsection 2.3 for solving analogous perceptual reasoning tasks.

**Model choice** We base our approach on the LongT5 encoder-decoder model [19], leveraging its extended context length to accommodate larger riddles. The T5 family was selected for its sequence-to-sequence capabilities, having been trained on a transformation task from non-causal to causal text [20]. This pre-training instills non-causal attention mechanisms within the encoder, making it well-suited for associative learning.

```

solve: train input1 2999 4299 4442 2922 output1 19 4 4 294 2999 4429 4492 2922.
input2 27757 27525 22277 57757 52257 output2 29 5 5 275 27757 27275 27257
55727 52257.
test tinput1 4884448 8844844 4844488 4848848 8888484 8448844 8848884
toutput1 55 7 7 48 4884448 8488884 4484448 4888448 8848484 8484844 8848884.

```

Figure 2: An example of the text prompt fed to our model. Here, each input and output grid is unrolled into a flat sequence of pixel-color values, which are then concatenated with keywords such as `train`, `test`, `input`, and `output`. The phrase `solve:` indicates that the model should produce the correct transformed grid (`toutput1`) corresponding to the given test grid (`tinput1`).

To fine-tune the model, we encode each riddle as a single text sequence, where grids are unrolled row-wise, with pixel colors represented numerically and rows separated by spaces. By presenting the complete riddle as a unified input, the model processes all grid-pairs simultaneously, allowing tokens to influence each other, shown in Figure 2. This aligns with the recommendations in Section 2.3.

**Direct output** This direct output methodology stands in contrast to some prior work that instead attempts to produce code as an intermediate output that can then be run to produce the output grid from the input [16]. While this has some benefits, such as being able to produce a solution that can be run and tested, it also has some drawbacks. Producing code that can solve the riddle is typically a much harder task than simply producing the output grid directly [21]. A code

based solution must be much more explicit and well defined, and it must be syntactically correct, working correctly with all the different possible input grids. A human solver would make a similar argument, preferring to just solve the riddle directly, and would typically require a much shorter time to do so, compared to coding a working program that solves the riddle.

This direct output approach stands in contrast to earlier methods that generate code as an intermediate step, which can then be executed to produce the solution grid [16, 22]. On the one hand, such code-based methods have the advantage of verifiability: running the program directly tests its correctness. On the other hand, producing code to solve a riddle is typically more difficult than simply generating the final grid. [21] specifying a concept or process in full detail often proves more demanding than acting out the game or task itself. A code-based solution must be thoroughly defined, syntactically valid, and capable of handling multiple potential input grids. By contrast, a human solver typically finds it more straightforward and faster to solve the riddle outright than to write, debug a general-purpose program that performs the same task.

Generating code as the final output does not fundamentally alter the broad dynamic search process through which LLMs solve riddles—this internal flexibility and reasoning remain essential. However, it does shift the abstraction space, training the model to handle perception and action via programming constructs instead of direct grid outputs. A notable benefit of code generation lies in its strong validation: one can run the generated program on the provided examples to confirm correctness. Yet, our experiments found that the added complexity of producing a syntactically correct, general-purpose solution introduced extra challenges and did not match the performance of direct output generation in initial trials.

### 3.1.2 Multi-task Training

Multi-task training compels the model to manage multiple modes and contexts simultaneously, thereby reducing its reliance on memorizing individual task details and nudging it toward genuine contextual reasoning [23]. In our setup, we integrate additional tasks requiring high levels of contextualization and reasoning—drawn from various NLP datasets—alongside the ARC data. This approach boosts ARC performance, mirroring findings by [23], who demonstrated that vanilla transformers can exhibit robust learning-to-learn capabilities when trained on a sufficiently large and varied set of tasks. Although [23] employed simpler permuted vision tasks rather than abstraction- or reasoning-based datasets, their conclusion that scaling task diversity helps escape the “memorization regime” remains consistent with our observations.

### 3.1.3 Code pre-training and contextualization

We observe that training on coding tasks offers a more pronounced performance boost on ARC than merely adding multi-task training derived from language or NLP domains. It is generally easier to continue a sentence halfway through a paragraph than a code file halfway. Code datasets inherently demand meticulous attention to detail and context, requiring the model to keep track of variables and resolve dependencies. This greater focus on accuracy and hierarchical reasoning means that memorization alone is insufficient—an important distinction from many NLP tasks where world knowledge and memorized associations play a larger role. As noted by [24], coding data “is more logical and less ambiguous” ultimately fostering a better focus on context.

A more complete discussion of recent literature supporting the use of code data in LLM training can be found in section 5.

### 3.1.4 Automatic Riddle Generators

Programmatic riddle generation is a valuable strategy to expand ARC training data and enhance model learning. To facilitate this, we employ Domain Specific Language (DSL) techniques, drawing inspiration from the work of Andreas Koepf [25] and Johan Sokrates Wind’s (Icecuber) DSL [26], to construct synthetic riddles by sampling function names and their parameters. A key aspect of our training approach involves training the model to infer these underlying DSL function names and parameters from the input riddle grids, in addition to predicting the final output grids. This dual-prediction strategy, where models learn to predict both the output grid and the DSL function names, contributes to more robust performance compared to training on either target alone. While DSL-generated data represents a portion of our overall training corpus, it serves to illustrate important data generation concepts we utilize.

To further diversify our ARC-specific training data, we also employ various more traditional riddle generators. These generators produce complete input-output pairs and test grids, enriching our training dataset. We observe that if riddle examples leave aspects of the transformation underspecified, the model may inadvertently learn to encode these ambiguities directly into its weights, rather than inferring them contextually. This can lead to undesirable biases and

reliance on memorization. To mitigate this, we deliberately err on the side of overspecification in our generated riddles, ensuring sufficient information for the model to unambiguously determine the intended solution.

We give a more detailed description of the synthetic riddle generators in Appendix A.

### 3.2 Solution Part 2: Optimizing in the Evaluation Loop (Test-Time Fine Tuning)

During evaluation, we leverage each test riddle’s demonstration examples to create synthetic training data. Specifically, we select a grid pair from the riddle’s provided examples and repurpose it as a new “test example” forming a new, smaller riddle. We have the answer to this new riddle, as it was taken from the demonstration examples, and so we can use that to train the model at test time. To get a lot more data and ensure this riddle differs from the original, we apply several augmentations:

- **Color permutation:** Randomly shuffle the color labels throughout the riddle.
- **Spatial transformations:** Rotate, flip, or transpose the input and output grids, sampling from the dihedral group  $D_4$ .
- **Shuffling:** Randomly reorder the input demonstration examples.

We then perform a brief round of fine-tuning on these augmented riddles before generating predictions for the test grids. This procedure can be seen as a form of test-time training [27] and is referred to here as *Test-Time Fine Tuning* (TTFT).

#### 3.2.1 Motivation for TTFT Through Iterative Reframing

The central idea behind Test-Time Fine Tuning (TTFT) is that the solver may initially make mistakes on the private test set, and we can exploit that feedback to refine its approach. Much like a human solver, the model can re-evaluate and iterate on potential solutions. For instance, consider the scenario illustrated in Figure 3, where the correct transformation is to select the color of the line that does not intersect others.

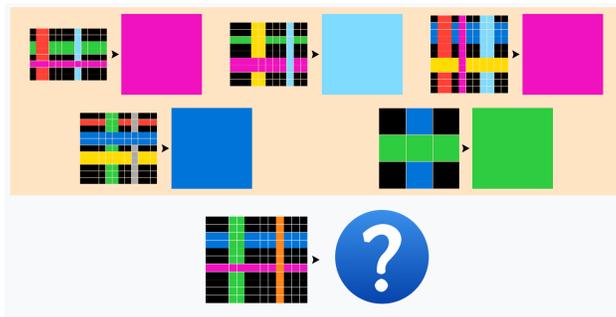


Figure 3: An example of a simpler ARC riddle.

A solver or human might instead begin by hypothesizing that the intended solution is to choose the color of the *thinnest* line, perhaps because the first few examples happen to support that interpretation.

This kind of oversight often arises from limited attention or inadequate depth of processing: the model (or a human) fails to fully observe all relevant information and thereby locks into a flawed “framing.” Once the solver is committed to an incorrect framing (for example, consistently searching for the thinnest line), it will have to continue to discard potentially crucial data and remain blind to the correct pattern. The input grids have already been processed with the incorrect framing/bias. In such a case, only a complete reprocessing of the grids—where the correct framing is established from the outset—can guide the solver to correctly infer the solution.

Proper framing is consistently shown to be critical for perceptual models, greatly influencing performance. One illustration of this appears in [28], which observes that language models often ignore information in the middle of a prompt, yet perform significantly better when the framing (in the form of the question or key instruction) appears at both the beginning and end of the prompt. Another relevant example is instruction tuning in LLMs [29, 30], wherein models are trained to adopt a “helpful” framing instead of a purely next-word prediction mode.

Hence, mechanisms for “reframing” are crucial in solving perceptually rich tasks, especially in the ARC setting, where each riddle’s solution demands a tailored framing. TTFT provides a means to adapt these frames based on feedback derived from newly generated training data—mirroring the human process of iteratively revisiting and adjusting hypotheses until the training examples are correctly solved, before finally addressing the test grids.

**Why take full parameter update steps (Full fine tuning)** Although several lighter-weight alternatives exist for adapting models to downstream tasks—including chain-of-thought prompting [31], few-shot prompting [32], and low-rank adaptation methods [33]—we chose full parameter updates primarily due to simplicity and reliability. Training the model using full parameter updates is naturally powerful enough to generate the needed abstractions. Given ARC’s

demand for generating diverse and genuinely new abstractions at test-time, we opted for this straightforward, guaranteed method of update, even though other adaptation techniques may also offer promising results and sufficient updating power.

### 3.2.2 Attention and masking

We specifically choose encoder-decoder architectures because they incorporate non-causal (unmasked) attention within the encoder, allowing each token to simultaneously attend to the entire input sequence. This capability is critical for enabling the model to fully interpret and contextualize ARC riddles from the outset.

By contrast, if the riddle were presented using causal (masked) attention, tokens appearing earlier in the sequence would not have access to the complete context, preventing them from forming accurate early-stage representations or hypotheses simply due to lack of available information. Tokens representing input grids would be unable to attend forward to their corresponding output grids, significantly limiting the model's reasoning about intended transformations. To validate the practical importance of this non-causal attention mechanism, we experimentally compared our encoder-decoder approach against similarly sized causal decoder-only models and found that the encoder-decoder structure yielded substantially better performance. We were not able to run experiments to disambiguate whether this is due to the non-causal attention masking or the encoder-decoder architecture itself, but it's likely that the non-causal attention is the main factor here.

### 3.2.3 Specialization to the riddle

Test-time fine-tuning can also enhance the precision required to produce completely accurate outputs. Even when the model correctly identifies the transformation function, minor execution errors—such as inaccuracies of a pixel or two—may occur. These errors are likely due to limited model depth or capacity, restricting the model's ability to execute transformations perfectly on the first attempt. TTFT can mitigate these issues by adapting the model specifically to the current riddle, refining its “execution” capabilities, and enabling it to achieve precise, pixel-perfect outputs.

### 3.2.4 Beam Search for Solution Space decoding

Because our model generates output grids autoregressively, a purely greedy decoding strategy is brittle: even a single incorrect token leads to an unrecoverable trajectory. Beam search [34] addresses this by maintaining multiple candidate solutions simultaneously, pruning all but the most promising branches at each decoding step according to cumulative probabilities.

This strategy allows the solver to effectively handle cases where the correct next token may initially have a lower probability but becomes clearer in subsequent steps. Conversely, incorrect trajectories naturally lose confidence as they proceed; a well-calibrated model will assign increasingly uniform probabilities across candidate tokens when uncertain, causing these erroneous paths to be rapidly discarded. Although models can occasionally become *confidently* wrong, beam search generally remains beneficial [34], and the ARC dataset in particular reaps substantial advantages from this capability: each riddle has precisely one correct solution, amplifying the divergence between correct and incorrect paths under beam search.

## 3.3 Augment, Inference, Reverse augmentation and Vote (AIRV)

We propose a test-time augmentation strategy called *Augment, Inference, Reverse-Augmentation, and Vote (AIRV)*. The procedure begins by applying a spatial transformation to the input riddle (e.g., rotation or flipping). We then get predictions on the transformed riddle to obtain a predicted output grid, which is subsequently *reversed* back to the original orientation. Finally, we gather multiple such predictions from different spatial augmentations and use a voting scheme to select the most frequent (or most confident) output grid.

Unlike beam search [34] or temperature sampling [35], AIRV can generate duplicate predictions (after reversing). This enables a voting mechanism that amplifies strong, consistent solutions and filters out noisy variants. This is particularly useful in the ARC setting, where each riddle has only one correct answer. Assuming a somewhat competent model, a voting mechanism then provides a very effective way to make salient the more dominant and consistent grid predictions (the more dominant solution ideas) from other more noisy predictions. In our opinion, this works because given a reasonably performant model, there are many ways that solutions can be incorrect, while there is only exactly one correct solution. This can be seen as analogous to the clustering step in the AlphaCode methodology [36] where the generated programs are clustered and the most common program cluster is selected as a proxy for most likely correct program.

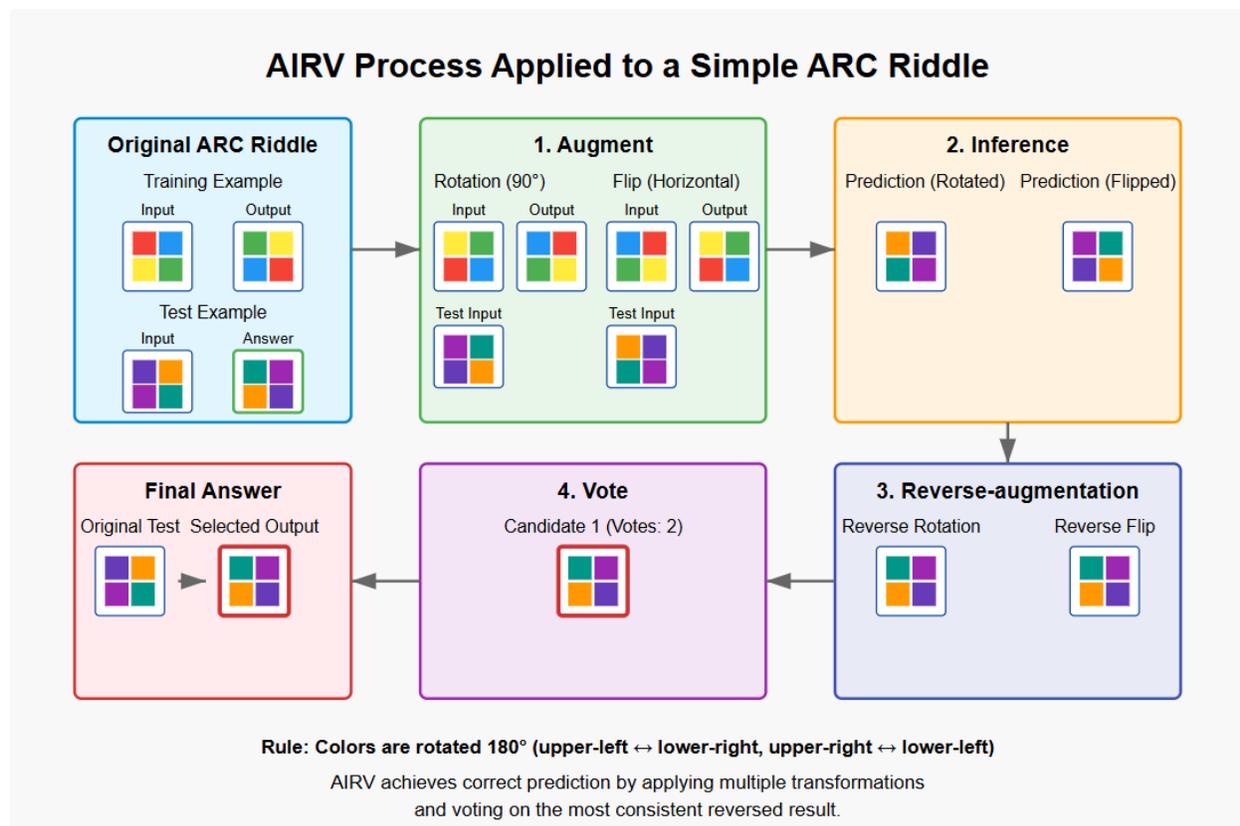


Figure 4: **AIRV process applied to a simple ARC riddle.** Starting from the original riddle (blue panel), the pipeline (1) *Augments* the grids via rotations and flips, (2) *runs inference* on each transformed instance, (3) *reverses* every prediction back to the original frame of reference, and finally (4) *votes* on the most consistent output.

## 4 Results

### 4.1 ARC dataset splits

The ARC dataset consists of 400 training riddles, 400 public evaluation riddles, and 100 private evaluation riddles that are not accessible to the public [37]. The training set riddles are the easiest, then public evaluation riddles are harder, and finally the private test set have been shown to be harder than the public evaluation set [38].

#### 4.1.1 Testing setup

We report our results on the private test set, with the following test-time compute limitations imposed by the competition compute environments available [7, 37]: Namely 2 hours of runtime on a single P100 GPU (16 GB VRAM).

Accuracy between predicted output grids and the ground truth output grids is measured by only counting exact matches over the whole predicted grid, allowing for two grid attempts per task (top-2).

### 4.2 Analysis and Discussion

To evaluate the effectiveness of our methodology, we compare the following configurations:

- **Zero Shot (No TTFT/AIRV):** Direct prediction using the ARC-trained LongT5 model.
- **AIRV Only:** Applying the AIRV technique (with beam search decoding) but without TTFT.
- **TTFT + AIRV:** Combined use of TTFT and AIRV (with beam search decoding).

We also train small and Large LongT5 variants on our training data. Due to pre-training compute constraints, only train those models on around 10% of the total training data.

Long T5 Base Model (Full Training)

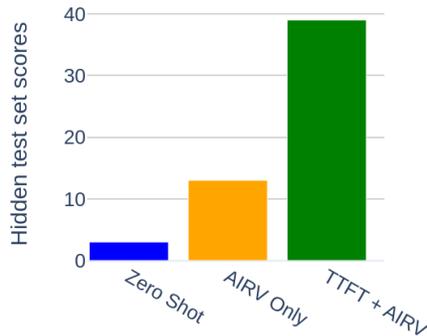


Figure 5: Results for the fully trained base model in the 3 different test-time configurations.

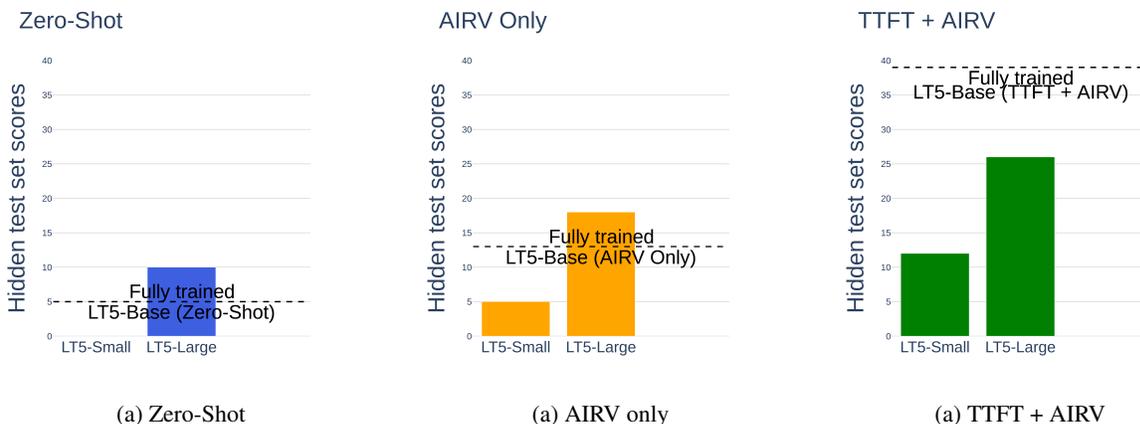


Figure 7: Performance on the ARC’s private test set. Each subfigure analyzes the impact of the different configurations of the test time techniques introduced. The small and large models trained on a subset of the training data are compared to the base model trained on the full training data. The effect of increasing model size is contrasted with the effect of increasing the number of training examples across the different test time techniques. We see that, model size has a significant impact despite a significantly reduced training set, except for when TTFT implemented at test time.

| Model                          | Zero Shot (No TTFT/AIRV) | AIRV Only | TTFT + AIRV     |
|--------------------------------|--------------------------|-----------|-----------------|
| Fully trained Base LongT5      | 5                        | 13        | 39 <sup>1</sup> |
| Partially trained small LongT5 | 0                        | 5         | 12              |
| Partially trained large LongT5 | 10                       | 18        | 26              |

**AIRV:** Augmentation and voting also enable higher performance. Absolute AIRV gains show positive scaling with model size. AIRV alone can gain up to 260% (in the base model, fully trained scenario).

**TTFT:** Test-time fine-tuning significantly increases the model’s score, results are consistent across model sizes and training regimes. Performing TTFT before running inference with AIRV leads to an additional 300% gain in performance.

An early version of this approach achieved first place in the 2023 ARCathon [40]. A version of this approach with more optimizations and extended ARC training, achieved the highest score on the ARC-AGI private test set in 2024 (58%) [39].

<sup>1</sup>A version of this approach with more optimizations and extended ARC training, achieved the highest score on the ARC-AGI private test set in 2024 (58%) [39].

**Experiments on model size** The Large model gets a lower score on TTFT but a higher score on Zero Shot and AIRV only, even with the lighter training. We see this trend holding across small, base and large models, where zero shot and AIRV only performance trends with size of the model. This scaling behavior is typical in deep learning [41, 42], but in this testing regime (ARC), can be explained by the fact that the larger the model the bigger and more expressive forward pass will be (more, and wider layers means more associations that can be made in the forward pass). This possibly accounts for the increased performance in the AIRV and zero shot setting, aligning well with the motivation in subsection 3.1.

Benefits of increased pre-training on the ARC dataset did not beat the benefits of simply increasing the model size. This perhaps indicates that the forward pass flexibility of the models is impacted much more by the model size compared to pre-training. This aligns with general theory regarding model scaling laws and how they impact typical reasoning benchmarks [42, 12].

**The effect of pre-training on TTFT** Interestingly, while increased pre-training does not seem to beat the effect of simply increasing the model size, when it comes to zero shot and AIRV only performance of these models, it does significantly improve score in the TTFT + AIRV setting compared to the base model. This effect is not likely only due to that larger models take more time to train (and TTFT) as the base model still sees a much larger boost in performance (300%) from TTFT than *both* large and small models (140% and 240% respectively). We discuss why this may be in section 4.2.

**Contextualization at Test Time vs. Pre-Training on ARC Riddles** We have motivated why high quality contextualization is a crucial element for tackling the ARC dataset. Yet, our experimentation shows that substantial pre-training on ARC riddles remains indispensable for achieving state-of-the-art performance with TTFT, culminating in our 2024 highest score on ARC-AGI. While the space of possible ARC transformations is vast, pre-training does not merely “leak” memorized solutions. Instead, it imparts both the foundational “core knowledge priors” described by [1] and a range of *more subtle but highly important* priors. By this, we refer to heuristics such as a preference for simpler, human preferred transformations (the “simple/simpler transformation” bias), a drive to validate transformation hypotheses across examples (the “looking for confirmation” bias), and even the basic notion that each example is formed by a paired input and output grid. Without these biases deeply embedded in the model weights, the solver would be far less efficient in forming or testing hypotheses during inference, and a lot of the forward pass would be spent at just merely realizing these basic things about the problem setup.

This is further supported by the following recent findings. [43, 44] have demonstrated that predictive features emerge at different layers during pre-training, with simpler but equally predictive features appearing earlier in the network. Recently, [45] also show that longer pre-training allows for complex but predictive representations to “sediment” (move into the earlier layers). We hypothesize that the extensive ARC pre-training allows for more “room” for test time features to emerge (during test-time fine-tuning), because the base arc priors have sufficiently sedimented into the very early layers. This sedimentation process, may be the crucial key for enabling the model to handle more complex, task-specific reasoning when test-time fine-tuned on unseen riddles.

**Contrasting pre-training with program synthesis** These priors are particularly relevant when considering a solver that is based on program synthesis. Program-synthesis-based approaches explicitly encode pair association and transformation-confirmation heuristics by searching for a program that correctly transforms input grids into their corresponding output. While these methods avoids the need for extensive domain-specific pre-training, it still requires significant human intervention to guide the program synthesis algorithm. Specifically, humans must direct the algorithm to search for transformations that align input grids with output grids and ensure that the transformations are correct using the other grid-pairs. Moreover, program synthesis solvers, with their manually encoded heuristics, are generally less effective when faced with perceptual problems that involve an almost limitless range of possible transformations and framings.

These are important considerations to keep in mind when considering the trade-offs between extensive pre-training compute and why its necessary, and the explicit program synthesis approach.

## 5 Related work

Classically, meta-learning can be regarded as a strategy to automate model design or parameter selection across numerous tasks, often formulated as a two-level optimization problem. In such setups, an “outer” model accumulates meta-knowledge, while an “inner” model adapts rapidly to each new task. More recent advances in *in-context learning* (ICL) have sidestepped explicit inner-outer distinctions, instead relying on the model’s forward pass to perform meta-learning. This behavior is commonly observed in transformer architectures trained on data with specific distributional

properties [46], enabling impressive performance on various tasks. ICL appears to support a more data-efficient form of meta-learning, one capable of storing and leveraging priors in a flexible way—especially appealing for applications like ARC.

A notable illustration of forward-pass ICL exhibiting strong generalization is the MLC architecture [47]. MLC mines for examples similar to the new task on their dataset, and, similar to our methodology, combines all relevant examples into the model's forward pass at once, allowing it to function as a meta-learner *within* the forward pass. This design substantially improves performance and generalization, highlighting the potential of ICL-based approaches for tasks that require complex reasoning.

## 5.1 Concurrent Work Building on TTFT

An interesting replication that is based on our proposed Test-Time Fine Tuning (TTFT) approach can be found in the work of [48], who explore the idea of adapting models on-the-fly for ARC tasks. Their method explicitly incorporates a similar short fine-tuning phase during inference on ARC, closely mirroring our TTFT paradigm. This aligns with our findings that dynamic adjustments at evaluation can significantly enhance performance, especially when encountering tasks requiring newly discovered transformations or abstractions. Also based on [49, 50] their methodology and results strongly corroborate and align with ours.

Building further upon our proposed TTFT approach, [51] recently explored the interplay between inductive and transductive reasoning specifically within the ARC domain. Their study trains neural networks on synthetic datasets generated from Python-based implementations of ARC transformations, based on [49], they also use TTFT for their transductive domain. Their experiments highlight that inductive program synthesis excels in precise symbolic tasks, whereas transduction demonstrates strength in more perceptually oriented problems. By effectively ensembling these complementary models, their approach achieves strong results, strongly validating the effectiveness and flexibility of TTFT-based adaptation to achieve high performance.

## 5.2 Code data in LLM training

Emphasizing coding and code training in LLMs is not new. Coding datasets form a significant part of LLM pre-training corpora, even in non-coding based models [52, 53]. It is correlated with improved performance on reasoning tasks, [54] find that code based models consistently outperform text based models in reasoning, even on synthetic reasoning tasks formulated as natural text. Further, [24] show that pre-training LLMs with a mix of text and code increases the general reasoning capability of the model. They also show that code at the instruction tuning stage enhances task specific reasoning. [55] also show that code models, even when outputting text, outperform text models on few shot structured reasoning evaluations. More recently, [56] carefully ablate the effects of code-data in pre-training and find positive effects on compositional tasks like semantic parsing and mathematics.

## 5.3 ARC dataset's related work

The Abstraction and Reasoning Corpus (ARC) dataset [1] presents a significant challenge for artificial intelligence systems due to its emphasis on reasoning from minimal examples. Numerous approaches have been proposed to tackle ARC tasks, ranging from leveraging LLMs to developing specialized neural architectures and neuro-symbolic methods to brute force search based methods.

### 5.3.1 Evaluating LLMs on ARC Tasks

Several studies have explored the capabilities of LLMs on ARC tasks without additional training. Mirchandani et al.[57] investigated whether LLMs can act as general pattern machines by providing the entire ARC task as context to GPT-4 and GPT-3.5. They achieved an accuracy of 10.6% on the combined ARC dataset and 6.75% on the public test set, indicating limited performance. Similarly, Mitchell et al.[58] compared GPT-4 and GPT-4V to human performance on a simplified version of ARC called ConceptARC [59], finding that GPT-4V did not significantly improve performance over GPT-4 and that both models underperformed compared to humans.

Other works have attempted to improve LLM performance by altering input representations. [60] translated visual ARC tasks into textual descriptions to leverage LLMs' reasoning capabilities, achieving 20% accuracy with GPT-3 on the ARC training set. [61] emphasize the importance of object-based representations, introducing an object-centric encoding to feed into LLMs. They tested GPT-4 on the easiest 50 ARC tasks and solved 23 out of 50.

These studies highlight that frozen LLMs possess some pattern recognition abilities and may possess some associative learning abilities out of the box in their pretrained forward pass, but they struggle with the abstraction and reasoning required for ARC tasks.

### 5.3.2 Neuro-Symbolic and Program Synthesis Approaches

Another line of research focuses on combining neural networks with symbolic reasoning or program synthesis to solve ARC tasks. Wang et al.[62] proposed a method where GPT-4 generates high-level textual hypotheses about the task, then translates these into code to solve the task. Testing on a subset of 40 ARC training tasks, they achieved a success rate of 27.5%, which increased to 37.5% with human selection of correct hypotheses. However, this again relies heavily on the frozen LLM's forward pass to be powerful enough to do the reasoning required for ARC tasks.

[63] introduced CodeIt, a method that generates code based on grid-pairs and uses a self-improving loop during evaluation. CodeIt solves 14.8% of the ARC evaluation tasks and runs into the limitations we describe in 2.3.

[64] developed Generalized Planning for ARC (GPAR), modeling ARC tasks as generalized planning problems in the Planning Domain Definition Language (PDDL) coupled with external functions representing object-centric abstractions of the grids. [64] achieved 50% accuracy on a subset of object-centric tasks. [22] used a dreamcoder inspired approach [65] with a domain-specific language for ARC tasks, achieving 18 out of 400 tasks on the ARC evaluation set.

These highlighted approaches attempt to incorporate a more symbolic reasoning approach to solve ARC tasks yet can only achieve a limited success rate or work only within a specialized domain. This may be due to the limitations of perceptual reasoning with these symbolic approaches, or may also be in some cases due to the added complexity of fully representing ARC transformations in code or domain-specific languages.

[14] utilized neural embeddings and vector arithmetic to solve ARC visual analogies but achieved only 2% accuracy on the public evaluation set. We discussed this weaknesses of this approach further in 2.3.

### 5.3.3 Brute Force Search

Icecuber [26] attempts to solve ARC by performing a brute force search over unary functions on pieces of input grids, forming a directed acyclic graph (DAG) of many possible transformations, until a DAG that creates the training output grid is found. This method won the 2020 ARC challenge competition on Kaggle [7].

### 5.3.4 Other Datasets

Other works have focused on simplified versions of ARC. [66] state that ARC is impenetrable for the time being and introduce the Sort-of-ARC dataset, which is only limited to 20x20 grids and only contains 3 objects with a limited set of transformations only. They emphasize object centric reasoning by using a controller to generate a solution vector, use slot attention transformer to extract object vectors, then update the solution and object vectors in an object centric way before generating the final solution using a spatial decoder on the result. We believe that a purely object centric approach does not generalize to tasks where objects are ambiguous, and the correct "object" is very task specific. [66] achieved 59% accuracy on out-of-distribution tasks in the Sort-of-ARC dataset.

[61] introduce the 1D ARC dataset, which is a simplified version of ARC with only one dimension. They emphasize the importance of object centric input representations and prompt GPT-4 with their object centric representations of the tasks. They state that they strategically select the easiest 50 tasks out of the training set and solve 23 out of the 50 tasks.

### 5.3.5 Summary and limitations of related work

To summarize, ARC has indeed proven to be a challenging problem for the current paradigm of AI, with state-of-the-art results remaining low on the private test set compared to other datasets in the field, despite 3 recent competitions on the ARC [7, 40].

We identify the following limitations of previous approaches:

- **Comparisons without computational constraints:** Some studies compare their score to others without reporting the computational cost of their methods, making it difficult to make an apples to apples comparison. We rather rely on private test set performance and the computational constraints set by kaggle to avoid this issue. This is a major weakness of related work, we find that our method is much more compute efficient while still achieving state-of-the-art.
- **Limited Generalization:** Many methods perform well on subsets of ARC tasks or simplified versions but fail to generalize across the full spectrum of ARC tasks.

- **Lack of Contextual Reasoning:** Approaches that process grid-pairs in isolation, such as CodeIt [63] and [14] and GPAR [64] struggle with tasks that require understanding relationships across multiple examples.
- **Lack of focus on perceptual reasoning:** Methods that do not focus on perceptual reasoning seem to face difficulties due to the complexity of searching through an almost infinite space of possible transformations.
- **Evaluation on Public Data:** Some studies evaluate their models on the public ARC dataset, which may have been exposed in pre-training data, potentially inflating performance estimates.

## 6 Conclusion

In contrast to previous work, our approach leverages LLMs with a focus on tackling arc with broader perceptual reasoning as the core component. We achieve this by both training a performant perceptual reasoner using existing state-of-the-art contextualizing models (LLMs). And secondly, by fine-tuning this perceptual reasoner on ARC tasks for each task at test time. Unlike previous methods we prove out our approach by evaluating on the full private test set, ensuring that our methodology generalizes well to unseen tasks. We achieve state-of-the-art performance on the ARC-AGI private test set while being much more computationally efficient than existing methods.

## Acknowledgments

This research was conducted with the help of the Google TPU Research Cloud (TRC) program. We would like to thank the TRC program for providing the TPU resources and for their continued support.

## References

- [1] François Chollet. On the measure of intelligence. *ArXiv*, abs/1911.01547, 2019.
- [2] Lucas Beyer, Olivier J. H'enaiff, Alexander Kolesnikov, Xiaohua Zhai, and Aäron van den Oord. Are we done with imagenet? *ArXiv*, abs/2006.07159, 2020.
- [3] Arsenii Kirillovich Moskvichev, Victor Vikram Odouard, and Melanie Mitchell. The conceptARC benchmark: Evaluating understanding and generalization in the ARC domain. *Transactions on Machine Learning Research*, 2023.
- [4] Suvir Mirchandani, Fei Xia, Pete Florence, brian ichter, Danny Driess, Montserrat Gonzalez Arenas, Kanishka Rao, Dorsa Sadigh, and Andy Zeng. Large language models as general pattern machines. In *7th Annual Conference on Robot Learning*, 2023.
- [5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics*, 2019.
- [6] 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. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.
- [7] François Chollet, Katherine Tong, Walter Reade, and Julia Elliott. Abstraction and reasoning challenge. <https://kaggle.com/competitions/abstraction-and-reasoning-challenge>, 2020. Kaggle.
- [8] Aniruddh Raghu, Maithra Raghu, Samy Bengio, and Oriol Vinyals. Rapid learning or feature reuse? towards understanding the effectiveness of maml. In *International Conference on Learning Representations*, 2020.
- [9] Mingzhang Yin, George Tucker, Mingyuan Zhou, Sergey Levine, and Chelsea Finn. Meta-learning without memorization. In *International Conference on Learning Representations*, 2020.
- [10] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, koray kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.
- [11] Shell Xu Hu, Da Li, Jan Stühmer, Minyoung Kim, and Timothy M. Hospedales. Pushing the limits of simple pipelines for few-shot learning: External data and fine-tuning make a difference. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9058–9067, 2022.

- [12] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D 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 Ziegler, Jeffrey Wu, Clemens Winter, Chris 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 H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.
- [13] Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. What learning algorithm is in-context learning? investigations with linear models. In *The Eleventh International Conference on Learning Representations*, 2023.
- [14] Luca H. Thoms, Karel A. Veldkamp, Hannes Rosenbusch, and Claire E. Stevenson. Solving arc visual analogies with neural embeddings and vector arithmetic: A generalized method. *ArXiv*, abs/2311.08083, 2023.
- [15] Jake Snell, Kevin Swersky, and Richard S. Zemel. Prototypical networks for few-shot learning. *CoRR*, abs/1703.05175, 2017.
- [16] Natasha Butt, Blazej Manczak, Auke Wiggers, Corrado Rainone, David Zhang, Michaël Defferrard, and Taco Cohen. Codeit: Self-improving language models with prioritized hindsight replay, 2024.
- [17] Charles Jin and Martin Rinard. Evidence of meaning in language models trained on programs, 2023.
- [18] Belinda Z. Li, Maxwell Nye, and Jacob Andreas. Implicit representations of meaning in neural language models. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1813–1827, Online, August 2021. Association for Computational Linguistics.
- [19] Mandy Guo, Joshua Ainslie, David C. Uthus, Santiago Ontañón, Jianmo Ni, Yun-Hsuan Sung, and Yinfei Yang. Longt5: Efficient text-to-text transformer for long sequences. *CoRR*, abs/2112.07916, 2021.
- [20] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683, 2019.
- [21] Jean Piaget. *The origins of intelligence in children*. The origins of intelligence in children. W W Norton & Co, New York, NY, US, 1952.
- [22] Mikel Bober-Irizar and Soumya Banerjee. Neural networks for abstraction and reasoning: Towards broad generalization in machines. *arXiv preprint arXiv:2402.03507*, 2024.
- [23] Louis Kirsch, James Harrison, Jascha Sohl-Dickstein, and Luke Metz. General-purpose in-context learning by meta-learning transformers. In *Sixth Workshop on Meta-Learning at the Conference on Neural Information Processing Systems*, 2022.
- [24] Yingwei Ma, Yue Liu, Yue Yu, Yuanliang Zhang, Yu Jiang, Changjian Wang, and Shanshan Li. At which training stage does code data help LLMs reasoning? In *The Twelfth International Conference on Learning Representations*, 2024.
- [25] A. Koepf. Arc research: Riddle synthesis, 2022. Software.
- [26] Icecuber Top-Quarks. Top-quarks/arc-solution: Code for 1st place solution to kaggle’s abstraction and reasoning challenge, 2020.
- [27] Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei A. Efros, and Moritz Hardt. Test-time training for out-of-distribution generalization, 2020.
- [28] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the Middle: How Language Models Use Long Contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 02 2024.
- [29] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022.
- [30] Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V Le. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*, 2022.

- [31] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.
- [32] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Nee-lakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris 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 H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.
- [33] Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.
- [34] Clara Meister, Ryan Cotterell, and Tim Vieira. If beam search is the answer, what was the question? In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2173–2185, Online, November 2020. Association for Computational Linguistics.
- [35] Matthew Renze. The effect of sampling temperature on problem solving in large language models. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 7346–7356, Miami, Florida, USA, November 2024. Association for Computational Linguistics.
- [36] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, 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.
- [37] Francois 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.
- [38] Mike Knoop. Introducing the arc-agi public leaderboard, Jun 2024.
- [39] Francois Chollet, Mike Knoop, Bryan Landers, Greg Kamradt, Hansueli Jud, Walter Reade, and Addison Howard. Arc prize website 2025. <https://arcprize.org>, 2025. ARC-AGI.
- [40] Arcathon winners 2023. <https://lab42.global/winners/>, 2023. Accessed: 2025-03-05.
- [41] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeff Wu, and Dario Amodei. Scaling laws for neural language models. *ArXiv*, abs/2001.08361, 2020.
- [42] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems, NIPS ’22*, Red Hook, NY, USA, 2024. Curran Associates Inc.
- [43] Katherine L Hermann, Hossein Mobahi, Thomas Fel, and Michael C Mozer. On the foundations of shortcut learning. *arXiv preprint arXiv:2310.16228*, 2023.
- [44] Harshay Shah, Kaustav Tamuly, Aditi Raghunathan, Prateek Jain, and Praneeth Netrapalli. The pitfalls of simplicity bias in neural networks. *Advances in Neural Information Processing Systems*, 33:9573–9585, 2020.
- [45] Thomas Fel, Louis Bethune, Andrew Kyle Lampinen, Thomas Serre, and Katherine Hermann. Understanding visual feature reliance through the lens of complexity, 2024.
- [46] Stephanie C.Y. Chan, Adam Santoro, Andrew Kyle Lampinen, Jane X Wang, Aaditya K Singh, Pierre Harvey Richemond, James McClelland, and Felix Hill. Data distributional properties drive emergent in-context learning in transformers. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.
- [47] B.M. Lake and M. Baroni. Human-like systematic generalization through a meta-learning neural network. *Nature*, 623:115–121, 2023.
- [48] Ekin Akyürek, Mehul Damani, Linlu Qiu, Han Guo, Yoon Kim, and Jacob Andreas. The surprising effectiveness of test-time training for abstract reasoning, 2024.

- [49] Jack Cole and Mohamed Osman. Dataset-induced meta-learning (and other tricks): Improving model efficiency on arc. <https://lab42.global/community-model-efficiency/>, 2023. Accessed: March 3, 2025.
- [50] Jack Cole, Mohamed Osman, Michael Hodel, Keith Duggar, and Tim Scarfe. Machine learning street talk. [https://www.youtube.com/watch?v=jSAT\\_RuJ\\_Cg](https://www.youtube.com/watch?v=jSAT_RuJ_Cg), 2024. Accessed: June 2024.
- [51] Wen-Ding Li, Keya Hu, Carter Larsen, Yuqing Wu, Simon Alford, Caleb Woo, Spencer M. Dunn, Hao Tang, Michelangelo Naim, Dat Nguyen, Wei-Long Zheng, Zenna Tavares, Yewen Pu, and Kevin Ellis. Combining induction and transduction for abstract reasoning, 2024.
- [52] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayanan Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.
- [53] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.
- [54] Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, Benjamin Newman, Binhang Yuan, Bobby Yan, Ce Zhang, Christian Alexander Cosgrove, Christopher D Manning, Christopher Re, Diana Acosta-Navas, Drew Arad Hudson, Eric Zelikman, Esin Durmus, Faisal Ladhak, Frieda Rong, Hongyu Ren, Huaxiu Yao, Jue WANG, Keshav Santhanam, Laurel Orr, Lucia Zheng, Mert Yuksekogul, Mirac Suzgun, Nathan Kim, Neel Guha, Niladri S. Chatterji, Omar Khattab, Peter Henderson, Qian Huang, Ryan Andrew Chi, Sang Michael Xie, Shibani Santurkar, Surya Ganguli, Tatsunori Hashimoto, Thomas Icard, Tianyi Zhang, Vishrav Chaudhary, William Wang, Xuechen Li, Yifan Mai, Yuhui Zhang, and Yuta Koreeda. Holistic evaluation of language models. *Transactions on Machine Learning Research*, 2023. Featured Certification, Expert Certification.
- [55] Aman Madaan, Shuyan Zhou, Uri Alon, Yiming Yang, and Graham Neubig. Language models of code are few-shot commonsense learners. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 1384–1403, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics.
- [56] Anonymous. How does code pretraining affect language model task performance? *Submitted to Transactions on Machine Learning Research*, 2024. Under review.
- [57] Suvir Mirchandani, Fei Xia, Pete Florence, brian ichter, Danny Driess, Montserrat Gonzalez Arenas, Kanishka Rao, Dorsa Sadigh, and Andy Zeng. Large language models as general pattern machines. In *7th Annual Conference on Robot Learning*, 2023.
- [58] Melanie Mitchell, Alessandro B. Palmarini, and Arsenii Kirillovich Moskvichev. Comparing humans, GPT-4, and GPT-4v on abstraction and reasoning tasks. In *AAAI 2024 Workshop on "Are Large Language Models Simply Causal Parrots?"*, 2023.
- [59] Arsenii Kirillovich Moskvichev, Victor Vikram Odouard, and Melanie Mitchell. The conceptARC benchmark: Evaluating understanding and generalization in the ARC domain. *Transactions on Machine Learning Research*, 2023.
- [60] G. Camposampiero, L. Houmard, B. Estermann, J. Mathys, and R. Wattenhofer. Abstract visual reasoning enabled by language. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2643–2647, Los Alamitos, CA, USA, jun 2023. IEEE Computer Society.
- [61] Yudong Xu, Wenhao Li, Pashootan Vaezipoor, Scott Sanner, and Elias Boutros Khalil. LLMs and the abstraction and reasoning corpus: Successes, failures, and the importance of object-based representations. *Transactions on Machine Learning Research*, 2024.
- [62] Ruocheng Wang, Eric Zelikman, Gabriel Poesia, Yewen Pu, Nick Haber, and Noah Goodman. Hypothesis search: Inductive reasoning with language models. In *The Twelfth International Conference on Learning Representations*, 2024.

- [63] Natasha Butt, Blazej Manczak, Auke Wiggers, Corrado Rainone, David Zhang, Michaël Defferrard, and Taco Cohen. Codeit: Self-improving language models with prioritized hindsight replay, 2024.
- [64] Chao Lei, Nir Lipovetzky, and Krista A. Ehinger. Generalized planning for the abstraction and reasoning corpus, 2024.
- [65] 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, page 835–850, New York, NY, USA, 2021. Association for Computing Machinery.
- [66] Rim Assouel, Pau Rodriguez, Perouz Taslakian, David Vazquez, and Yoshua Bengio. Object-centric compositional imagination for visual abstract reasoning. In *ICLR2022 Workshop on the Elements of Reasoning: Objects, Structure and Causality*, 2022.
- [67] S. Ouellette. Arc\_gym: A data generation framework for the abstraction & reasoning corpus, 2023. Software.
- [68] M. Hodel. Addressing the abstraction and reasoning corpus via procedural example generation, 2024. Preprint.
- [69] D. Hupkes, V. Dankers, M. Mul, and E. Bruni. Compositionality decomposed: How do neural networks generalise? *Journal of Artificial Intelligence Research*, 67:757–795, 2020.

## A Training Dataset Construction and Composition

Our approach to solving the Abstraction and Reasoning Corpus (ARC) Challenge incorporated a diverse set of training datasets, combining both existing public resources and custom-generated synthetic data. This data strategy was designed to develop robust abstract reasoning capabilities and improve grid-based pattern recognition tasks.

**Multimodal Grid Translation Datasets:** These datasets facilitate translation between visual and symbolic representations of grids - including Base64 images, text, English descriptions, and code implementations. They focus on simple shapes and positions to bridge visual and symbolic reasoning relevant to ARC.

**Extended PCFG Datasets:** Building upon the principles of Probabilistic Context-Free Grammars (PCFGs), we use an expanded PCFG dataset with 100 distinct string operations, designed to resemble ARC riddles. These datasets incorporate program synthesis components, emphasizing function name generation as part of the learning process.

**Cellular Automata and Mathematical Pattern Datasets:** We generate cellular automata tasks within the ARC framework and create riddle boards using mathematical equations. These datasets introduce varied pattern complexities and cross-item riddles that require learning underlying concepts across multiple examples. Repair-type riddles are also generated to induce priors related to fundamental visual reasoning concepts like symmetries, shapes, progression, and counting.

Our data generation strategy also leverages frameworks such as ARC\_gym [67] for supplementary examples with customizable grid characteristics. We also enhance existing datasets, including augmented ConceptARC data [3] and augmented official public ARC datasets. The recent integration of RE-ARC data [68], with its procedurally generated examples for the 400 ARC training tasks, has further improved performance by increasing exposure to task-specific patterns and transformations.

### A.1 Public Datasets

The training pipeline incorporated several established datasets from the research community:

#### A.1.1 Language and Reasoning Datasets

1. WizardLM Evolution Instruct V2 (196k examples)
2. TinyStories-GPT4 (150,000 examples)
3. SQuAD v2
4. Chain-of-Thought Submix
5. Open-Platypus
6. SlimOrca

#### A.1.2 Domain-Specific Scientific Datasets

1. Arxiv Math Instruct (50k examples)
2. Arxiv CS/ML Instruct (50k examples)
3. MathInstruct
4. Arxiv Physics Instruct (30k examples)

#### A.1.3 Multimodal and Visual Reasoning Datasets

We integrated several multimodal reasoning datasets from the M3IT collection, specifically focusing on:

1. CLEVR
2. NLVR
3. VCR
4. Visual-MRC

### A.1.4 Programming and Code-Related Datasets

1. Magicoder Evolution Instruct (110K examples)
2. Open Instruct v1 (derived from OASST and Dolly HHRLHF)

### A.2 Custom Synthetic Datasets

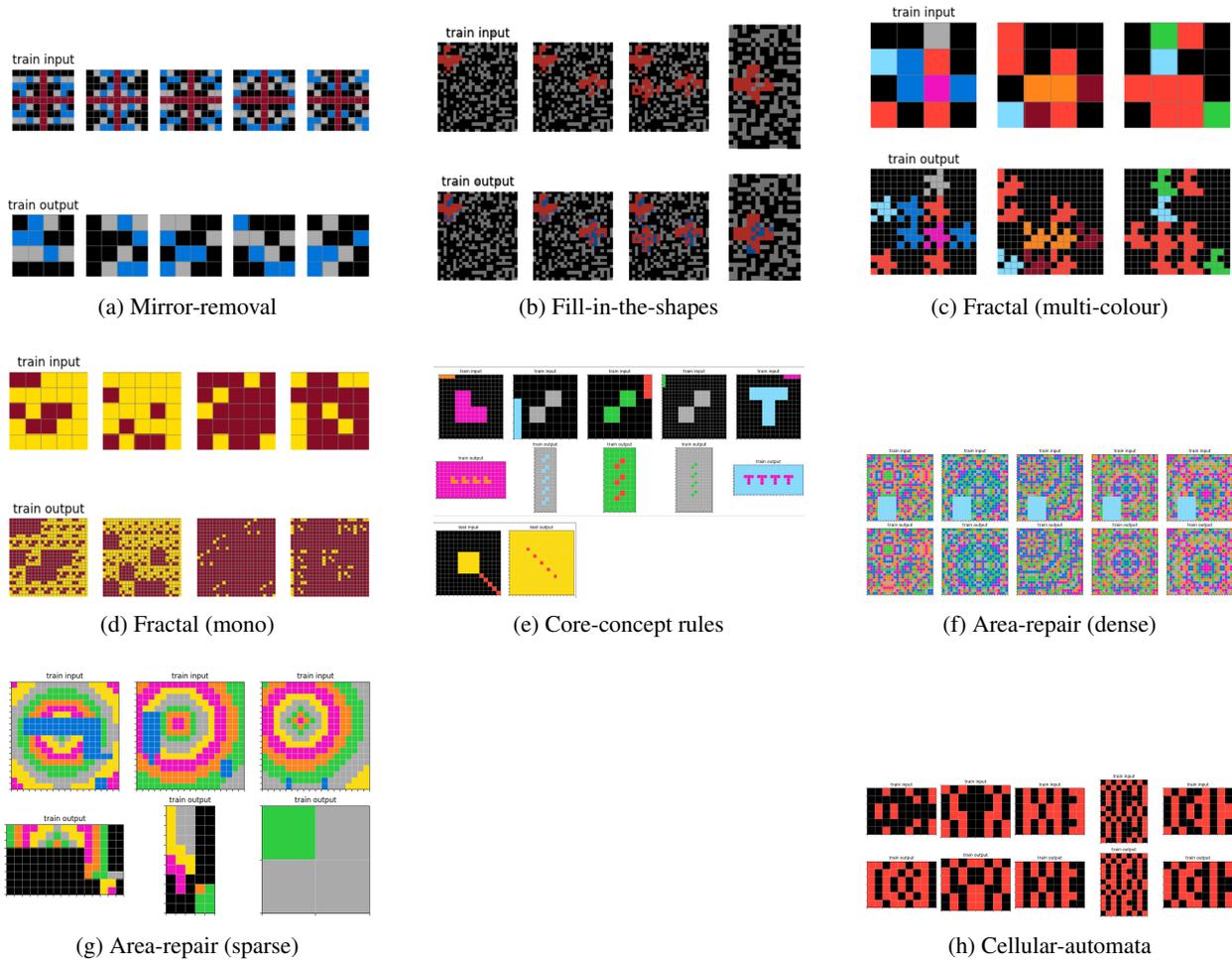


Figure 8: Gallery of synthetic ARC-style riddles used in pre-training: symmetry repair, noise-robust completion, fractal growth, core-concept generalization, area repair, and cellular-automata evolution with simple rules.

### 1. Arithmetic and Counting Tasks

We created fine-grained arithmetic and counting datasets that emphasized precise numerical operations and pattern recognition. These datasets were designed to strengthen the model’s capability in handling discrete mathematical operations commonly found in ARC tasks.

### 2. Multimodal Grid Translation

We developed a specialized dataset for translating between:

1. Base64 images of grids
2. Text representations
3. English descriptions
4. Code implementations

This dataset focused on simple shapes and their precise positions, creating a bridge between visual and symbolic reasoning.

### 3. Extended PCFG Dataset

Following work on neural networks' compositional capabilities [69] and the demonstrated effectiveness of Probabilistic Context-Free Grammars (PCFGs) for pattern learning [4], we significantly expanded our PCFG dataset to include:

1. 100 distinct string operations with few-shot examples
2. Tasks structured similarly to ARC riddles
3. Program synthesis components requiring function name generation

These tasks were generally designed to be more challenging than typical ARC problems, pushing the boundaries of abstract reasoning capabilities.

### 4. Multi-Task Integration Dataset

We created a synthetic multi-task dataset that combined multiple problem types within single training examples:

1. Arithmetic code generation
2. Chained boolean expression evaluation
3. PCFG tasks

The dataset was structured in JSON format, with both prompts and answers carefully formatted for consistency.

### 5. Cellular Automata and Mathematical Patterns

We incorporated additional pattern-generation approaches:

1. Cellular automata tasks within the ARC framework
2. ARC riddle boards generated using mathematical equations
3. Pattern complexity variations to induce diverse priors
4. Cross-item riddles requiring learning underlying concepts across multiple examples
5. Repair-type riddles for inducing priors including symmetries, shapes, progression, and counting

See Figure 8 for examples of synthetic ARC-style riddles used in pre-training.

## A.3 Data Sources and Generation

We incorporated several data sources and generation approaches:

### 1. Framework-Generated Data:

- (a) ARC\_gym framework [67] to generate supplementary training examples with:
  - i. Customizable grid sizes and complexity levels
  - ii. Controlled out-of-distribution task generation
  - iii. Variable numbers of primitives per task
  - iv. Systematic composition of basic operations
- (b) IceCuber-based riddle generation [25], including:
  - i. Synthetic riddle generation
  - ii. Solution graph prediction tasks

### 2. Enhanced Public Datasets:

- (a) Augmented ConceptARC data [3], leveraging:
  - i. Concept group organization
  - ii. Varying complexity levels
  - iii. Abstraction-focused variations
- (b) Augmented official public ARC datasets
- (c) Modified versions of publicly available ARC\_gym generated data

#### **A.4 Recent Additions**

In early 2024, we integrated data from RE-ARC [68], which provided procedurally generated examples for the 400 ARC training tasks. This addition contributed to improved performance through increased exposure to task-specific patterns and transformations.

#### **A.5 Training System Notes**

The training system utilizes prepared datasets in chunks for roughly 500,000 training examples. The chunks are needed to reduce CPU memory usage for processing. The order of the training examples is shuffled. It automatically incorporates new datasets without interrupting training and some datasets include instruction-following and chat-based interaction capabilities.