Combining Deep Learning Intuition and Domain-Specific Language in the ARC Challenge 2024

ARCade Warriors

Laurent Cheret lcher021@uottawa.ca

Emile Haas haas_emile@hotmail.com

Abstract

The Abstraction and Reasoning Corpus (ARC) is a difficult benchmark due to its resistance to memorization. Deep Learning approaches that directly map input grids to output grids become increasingly ineffective as the private test set introduces novel tasks that deviate from known patterns. Our contributions combine two key components. First, we develop an autoencoder model to generate abstract representations of individual grids, which are then aggregated into what we call "intuition vectors" of tasks. Through UMAP visualization, we demonstrate that these vectors capture meaningful semantic relationships between similar tasks, providing direction, or intuition for potential solutions. Second, we extend the established Domain-Specific Language (DSL) approach based on ARC's core knowledge priors, by introducing a Last-In-First-Out (LIFO) memory mechanism which we argue is fundamental for sequential reasoning by allowing an agent to store and retrieve intermediate states. We propose that solving ARC lies in a combination of collective human effort to generate different DSL paths, or solutions for a same task and using the abstract representations to guide a future deep learning approach to extend the current dictionary of solutions to new unseen cases.

1 Introduction: What's ARC and why is it difficult?

The Abstract Reasoning Corpus (ARC) [Cho19] is a collection of intelligence test problems designed to measure abstract reasoning capabilities. Each task consists of a set of input-output pairs of colored grid patterns, where the solver must infer the underlying transformation rules to correctly map new input grids to their expected outputs. Unlike traditional machine learning benchmarks, ARC explicitly tests for human-like reasoning abilities by requiring generalization beyond pattern matching or statistical regularities (memorization). The tasks incorporate core cognitive skills like object recognition, spatial relationships, arithmetic, and analogical thinking, while deliberately avoiding specialized knowledge or complex language understanding.

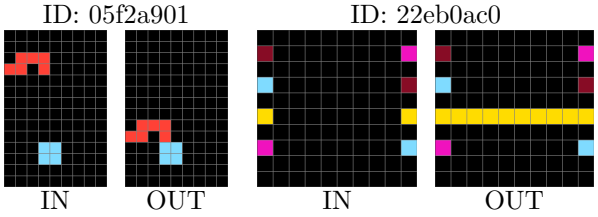

Figure 1: Two examples of ARC tasks with different objectives and reasoning. **05f2a901**: move red objects towards the blue endpoint. **22eb0ac0**: connect same color points only if they share the same row.

Each 2D grid in an ARC task can range in size from 1×1 (a single cell) up to 30×30, with dimensions and shapes dictated by the task's objective. Within a given task, output grids may differ in shape from their corresponding inputs, and grid shapes can also vary between input-output pairs, adding a significant layer of complexity. Each cell within these grids holds one of 10 possible integer values, from 0 to 9. Considering the full set of ARC grids, $\mathcal{X}$, yields an immense $10^{30\times30}$ possible

configurations, making the challenge of training a mapping $f : \mathcal{X} \to \mathcal{X}$ formidable—more daunting, perhaps, than finding a needle in a haystack. Another challenge lies in the limited dataset, with only 400 ARC tasks in the training set and 400 in the evaluation set, far fewer than what would be required to learn a reliable mapping f. Finally, ARC requires exact answers; any deviation from the expected output for a task is counted as incorrect, which means that we have to autonomously describe complex transformations, or programs to solve these tasks, in an exact manner. See Figure 1.

In the following work, we show how one could bypass these difficulties related to ARC. More specifically:

- We develop an ARC Grid Autoencoder, which enables us to represent tasks using the concept of "intuition vectors". We show through UMAP [MHM18], that similar intuition vectors translate to similar semantics for a significant number of ARC tasks analyzed.
- We propose our own DSL based on ARC core knowledge priors, adding the concept of memory operations such as "push" and "pop", which unlock the possibility of memorizing intermediate states of the path towards the solution. We argue that these operations are fundamental for a sequential approach using non-parametric functions.
- We also developped a Web Interface that allows users to play with our DSL and find solutions to available ARC tasks. Our future goal is to crowdsource the different solutions found by users to create a large dictionary of solutions through crowdsourcing, which we believe will play a key factor in the future solution of ARC. Available at `https://github.com/laurent-cheret/ARC.git`.

2 Prior Work

The ARC dataset, introduced in 2019 by François Chollet, serves as a new benchmark for assessing artificial intelligence beyond mere memorization and skill acquisition [Cho19]. While ARC presents a challenging set of tasks, humans generally perform well, achieving an average of 80% accuracy, demonstrating that the majority of tasks are solvable [JVLG21]. A 2024 study further classified ARC tasks as easy, medium, or hard, based on the percentage of participants who successfully solved them, revealing that while participants share similar goals, they often take diverse paths to solutions [LVLG24]. These observations inspired the development of sequential methods that leverage Domain-Specific Languages (DSL) [Hod23, Hod24]. These DSLs allow for the composition of simple building blocks in sequence to create more complex objectives, much like words forming sentences to convey intricate ideas. However, DSL approaches face limitations: they tend to be highly specialized yet constrained, leading to combinatorial explosions as the library of primitives grows to cover more tasks and objectives.

With the advent of Large Language Models (LLMs), new strategies emerged to tackle ARC. Chollet suggests that LLMs function as interpolative memory databases, capable of limited generalization due to model size and latent space dimensionality, but unable to achieve the broader generalization required for ARC [HVMR24]. Supporting this view, recent studies confirm that LLMs struggle with true reasoning tasks [VMSK23, WQR+24]. Current state-of-the-art LLMs achieve around 20% accuracy on ARC, which falls far short of results from top-ranking teams using more complex, multi-faceted solutions. Chollet posits that a successful approach to ARC may require a hybrid model, combining discrete program search with deep learning to provide the necessary guidance and intuition. In the following sections we will propose two main ideas for tackling the ARC challenge, the first one will cover our approach for creating abstract representations of tasks that can be extended to unseen examples, and the second one will be our DSL which implements additional memory operations.

3 An abstract representation of ARC tasks

Each ARC task consists of several input-output grid pairs, each 2D colored grid can be thought of as having 3 dimensions: height, width, and 1 dimension with 10 channels, so that $g \in \mathbb{R}^{9000}$. The examples pairs in an ARC task are intended to reveal an underlying transformation $\mathbf{T}$ that maps inputs to outputs. The goal is to apply this transformation to the test input to accurately generate the

test output. Thus, the core objective of this challenge is to identify **T** for each task in an automated manner. See Figure 2.

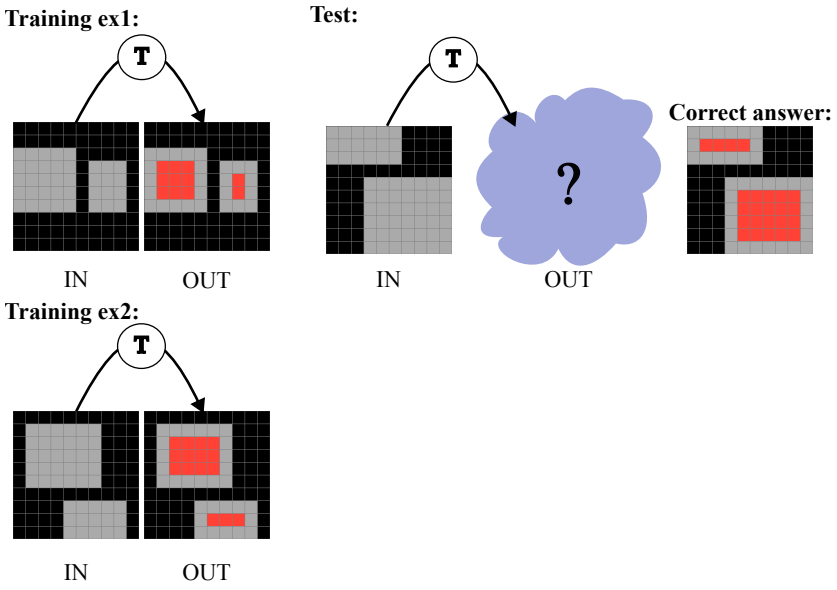

Figure 2: The same transformation **T** to be applied on all training examples, is applied in the test case, leading to the solution to that task. In this case for **bb43febb**, it can be roughly defined as: Fill grey rectangles with red, keep 1-cell borders.

However, working directly with colored grids is impractical, as **T** becomes complex to define explicitly. A more feasible approach is to use encoded representations of grids within a continuous space $\mathcal{Z}$, where the grid encodings are given as $z \in \mathbb{R}^n$ with $n < 9000$. In this space, transformations from input encoding z_i to output encoding z_o can be simplified as the difference between them, expressed as:

$$z = z_o - z_i.$$

For tasks with n input-output pairs, the average transformation of that task can be defined as:

$$Z = \frac{1}{n} \sum_{k=1}^{n} (z_{o,k} - z_{i,k}).$$

We call Z the intuition vector of a task. See Figure 3. We expect Z to reinforce the commonalities between all the training pairs of that task, giving us a average direction in $\mathcal{Z}$ which represents the complex solution of that task. It is thus easier to find tasks with similar intuition vectors in $\mathcal{Z}$ than it is to find in the space of grids $\mathcal{X}$.

3.1 From grids to intuition vectors

The limited size of the ARC dataset presents a considerable challenge for training deep learning models to encode entire tasks, as mapping from test input to a specific task output requires substantial information. Expanding this dataset would demand significant human effort for new ARC-like tasks since each one involves unique reasoning. However, rather than focusing on complete tasks, we could instead consider individual grids. This shift simplifies data augmentation, as no complex reasoning is needed to design a single grid, with a large pool of 10^{900} combinations. Additionally, rather than using deep learning models to map a single grid to its output, we believe their memorization power can be better leveraged for reconstruction. Autoencoders (AEs) [BKG21], a well-established model type, are especially effective at generating latent representations of inputs and handling unseen examples through Out-of-Sample Extrapolation (OSE).

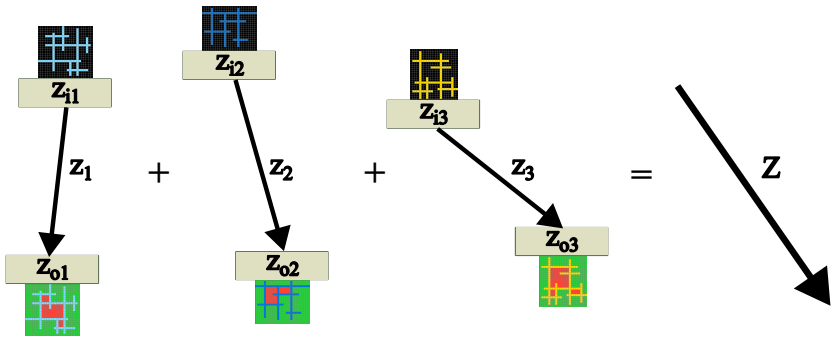

Figure 3: Example of intuition vector for task **7b6016b9**. The resulting intuition vector **Z** is the average of the intuition of all the training examples, in this case 3 examples. **Z** would reinforce the common transformations between the examples such as: "Paint background green", "Paint closed loops red", and give less importance to details such as the color of the enclosure (cyan, blue, yellow).

3.1.1 ARC Grid Auto-Encoder

Combining both ideas we develop an ARC Grid AE trained to reconstruct grids from the ARC training dataset which we also augment using a random grid generator. This architecture implements an autoencoder combining a transformer-based encoder with a deep neural network decoder. The encoder processes the input through a linear embedding layer (input_dim $\rightarrow$ 1024) followed by a transformer encoder with 16 attention heads and 3 transformer layers, using a feed-forward dimension of 1024. The decoder consists of sequential blocks, a final linear layer maps to the input dimension, followed by a sigmoid activation to produce the reconstruction. See Figure 4 for the architecture diagram.

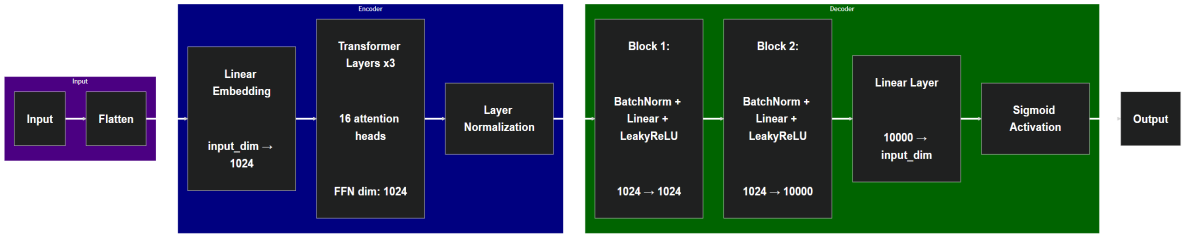

Figure 4: Diagram of the ARC Grid Auto-Encoder.

The training procedure implements a two-step optimization process for the autoencoder using mixed precision training. In each epoch, the model first updates the encoder parameters using a combination of reconstruction loss (measuring how well the input is reconstructed) L_{rec} and a distance-preserving loss $\mathcal{L}_{dist}$, weighted by a regularization parameter λ. $\mathcal{L}_{dist}$ works as a regularization metric to ensure the distances in the input space are preserved in the output of the encoder which helps to keep similar grids in similar regions of the manifold. Then, it separately updates the decoder parameters using only the reconstruction loss. Both parts use Adam optimization. This alternating optimization strategy allows the model to balance between learning good representations in the latent space and accurate reconstruction of the input data.

Encoder Loss:

$$\mathcal{L}_{\text{encoder}} = \mathcal{L}_{\text{rec}} + \lambda \mathcal{L}_{\text{dist}} \tag{1}$$

$$\mathcal{L}_{\text{rec}} = \frac{1}{N} \sum_{i=1}^{N} (x_i - \hat{x}_i)^2 \tag{2}$$

$$\mathcal{L}_{\text{dist}} = \frac{1}{N^2} \left\| \frac{D_x}{\max(D_x)} - \frac{D_z}{\max(D_z)} \right\|_2 \tag{3}$$

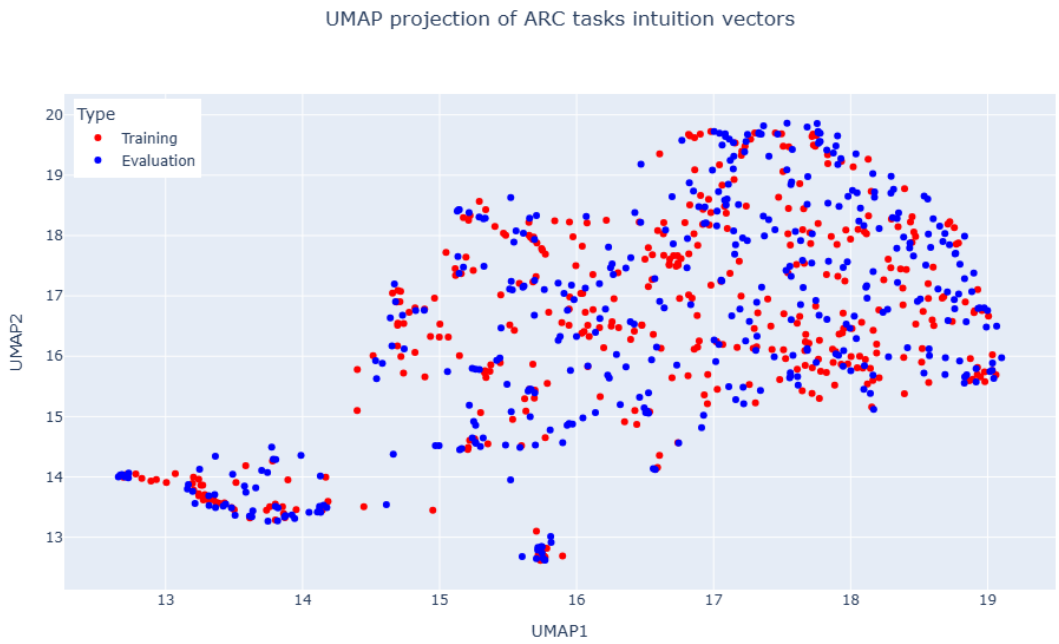

Figure 5: 2D UMAP projection of ARC tasks intuition vectors. Using Euclidean distance, $min\_dist = 0.0$, and $n\_neighbors = 20$.

$$D_x[i,j] = \|x_i - x_j\| \quad \text{(pairwise distances in input space)}$$
$$D_z[i,j] = \|z_i - z_j\| \quad \text{(pairwise distances in latent space)}$$

Decoder Loss:

$$\mathcal{L}_{\text{decoder}} = \mathcal{L}_{\text{rec}} \tag{4}$$

where:

- x_i, x_j are input samples
- $\hat{x}_i$ is the reconstructed sample
- z_i, z_j are latent representations
- $D_x, D_z \in \mathbb{R}^{N \times N}$ are pairwise distance matrices
- N is the batch size
- $\lambda \in \mathbb{R}$ is the regularization parameter

3.2 From intuition to similar ARC tasks

While developing an autoencoder for ARC grids is more straightforward than generating task solutions, it's crucial to validate that our intuition vectors capture meaningful semantic relationships. To assess this, we employ dimensionality reduction to visualize similarities between intuition vectors from both training and test sets. We chose Uniform Manifold Approximation and Projection (UMAP) [MHM18] for this analysis due to its efficiency with large datasets and its ability to preserve global data structure. We apply UMAP to project our dataset of 800 intuition vectors (400 each from training and test sets) from their original 1024-dimensional space to a 2D representation, allowing us to visualize and analyze the latent space structure. See Figure 5.

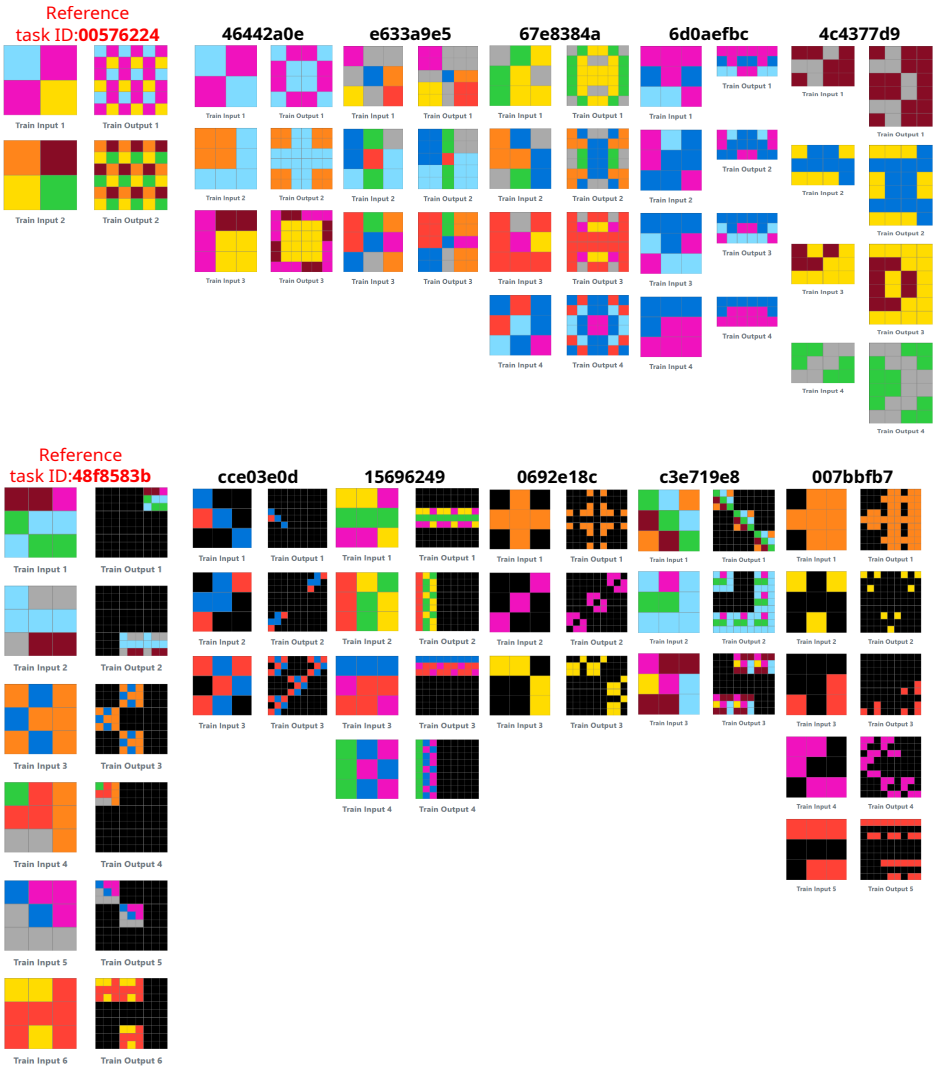

Figure 6: Tasks semantically similar to **00576224**, arranged by proximity: **46442a0e**, **e633a9e5**, **67e8384a**, **6d0aefbc**, **4c4377d9**. These tasks share a common theme of pattern replication. Similarly, tasks closest to **48f8583b** are: **cce03e0d**, **15696249**, **0692e18c**, **c3e719e8**, **007bbfb7**, all involving pattern replication with specific spatial constraints.

Using UMAP projection, we can identify tasks with similar abstract representations. As shown in Figure 6, task **00576224** clusters with five semantically related tasks that share pattern replication mechanics. Similarly, task **48f8583b** groups with five neighbors that specifically involve pattern replication with spatial positioning constraints. This clustering suggests that our intuition vectors successfully capture meaningful task similarities. We hypothesize that tasks with similar abstract representations will share similar solution structures in our DSL framework. This insight has several practical applications: it can guide the exploration and organization of the ARC dataset, inform the training of deep learning models, and assist in the design and refinement of DSL primitives based on identified task clusters. The latter will be explored in detail in subsequent sections.

4 A sequential approach to solving ARC tasks

While intuition can be represented as a direction vector in an abstract space $\mathbf{Z}$ pointing from initial state s_0 to final state s_f, the actual solution path rarely follows such a direct trajectory. Consider the analogy of traveling between two cities: while you can point directly to your destination, the practical journey often requires following established routes and using various modes of transportation rather

than walking in a straight line through buildings, forests, and natural obstacles.

Similarly, solving ARC tasks often requires breaking down complex transformations into manageable steps. Though theoretically possible to encode any solution as a single complex operation, humans naturally approach these problems sequentially, decomposing them into simpler, intermediate operations. This mirrors how a journey might combine different transportation methods (roads, railways, flights) with strategic waypoints, rather than attempting to traverse directly through obstacles.

In this analogy, programming each grid cell individually would be like taking small steps in a straight line – theoretically possible but inefficient. Instead, the Domain-Specific Language (DSL) approach provides high-level operations analogous to transportation infrastructure, offering efficient and reliable pathways to reach the solution through meaningful intermediate states.

4.1 Domain-Specific Language (DSL)

A DSL provides a specialized vocabulary of basic operations that can be combined to express complex solutions. While ARC deliberately avoids requiring real-world knowledge, its tasks are founded on fundamental cognitive priors – basic intuitions about objects, patterns, and transformations that humans naturally possess [Cho19]. The DSL approach leverages these priors by encoding them as elementary operations, allowing complex reasoning tasks to be decomposed into sequences of simple, interpretable steps.

4.1.1 Core Knowledge Priors

The Abstraction and Reasoning Corpus (ARC) is built upon fundamental cognitive priors that humans naturally possess. These core knowledge priors represent basic intuitions about how the world works and can be categorized into several key areas:

Object Permanence: The understanding that objects continue to exist even when they are not directly visible. In ARC, this manifests in tasks where objects maintain their identity across transformations or when partially obscured.

Spatio-temporal Continuity: Objects move along continuous trajectories and maintain their basic properties during movement. This prior is evident in ARC tasks involving pattern continuation, object movement, or spatial transformations.

Basic Geometry and Topology: Understanding of fundamental geometric concepts such as:

- Spatial relationships (above, below, adjacent)
- Basic shapes and their properties
- Rotations, reflections, and translations
- Connectivity and containment

Basic Arithmetic: Simple numerical concepts including:

- Counting and cardinality
- Basic operations (addition, subtraction)
- Pattern recognition in sequences
- Size and proportion relationships

Basic Physics: Intuitive understanding of:

- Object permanence and solidity
- Contact causality
- Support relationships
- Basic gravity-like behaviors

Compositionality: The understanding that complex objects or patterns can be broken down into simpler components, and conversely, simple elements can be combined to create more complex structures.

Abstraction: The ability to:

- Identify relevant features while ignoring irrelevant ones
- Recognize patterns at different levels of abstraction
- Apply learned rules to new situations
- Understand analogical relationships

We propose a set of primitive functions that map lists of grids to lists of grids. Each function preserves the input when the operation is invalid or undefined for the given input. These functions are organized into six fundamental groups: (1) **Basic Transformations**, including geometric (rotation, reflection), arithmetic (sum, subtraction), and logical operations (AND, OR, XOR); (2) **Object Operations** for manipulating grid elements through cropping, color modification, rotation, and collision handling; (3) **Abstract Operations** such as pattern generation and connectivity analysis; (4) **Color Operations** for color removal, swapping, and arithmetic; (5) **Critical Operations** handling list manipulation and state management; and (6) **Memory Operations** for state storage and retrieval, which will be discussed in detail later. See Figure 7.

In this example, an object operation is applied first taking us from s_i to s_1 by forming one grid per object of the input, in this case all three examples contain two objects so the resulting grid lists will contain two elements. $s_1 \rightarrow s_2$ represent a critical operation in which the order of the elements of the grid list is reorganized to put the largest objects found in a list in position 0. $s_2 \rightarrow s_3$ represents an abstract operation of collision, moving the object on grid in position 0 to the source in grid position 1 (OBS: If there are no sources, or no objects in the source then this primitive should do nothing and return the same input state s_2. Last step is a basic operation of addition by combining all grids from the list into a single one leading us to the desired output state s_f of task **05f2a901**.

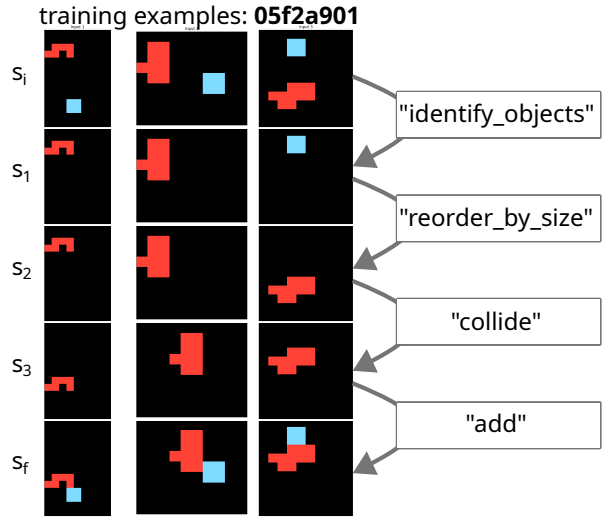

Figure 7: Example of sequence of steps needed in our DSL to go from the initial **05f2a901** state s_i to the final s_f, using four primitives. OBS: For every training example, in this case 3, we only see the grid occupying position 0 of its respective list.

4.1.2 Last-In-First-Out (LIFO) memory

A crucial addition to our DSL is the incorporation of memory operations, reflecting the fundamental role of intermediate state management in sequential reasoning. Taking inspiration from human problem-solving processes, where intermediate results are temporarily stored and later combined to construct complete solutions, we implement a Last-In-First-Out (LIFO) memory stack with three basic

operations: "push" (store current state), "paste" (retrieve last stored state), and "pop" (remove last stored state).

Figure 8 demonstrates this mechanism through task **4258a5f9**. While this particular example could theoretically be solved by enhancing the "draw_outlines" function to simultaneously generate outlines and grey centers, many ARC tasks inherently require multi-step solutions with state dependencies, making memory operations essential for reaching the final state s_f.

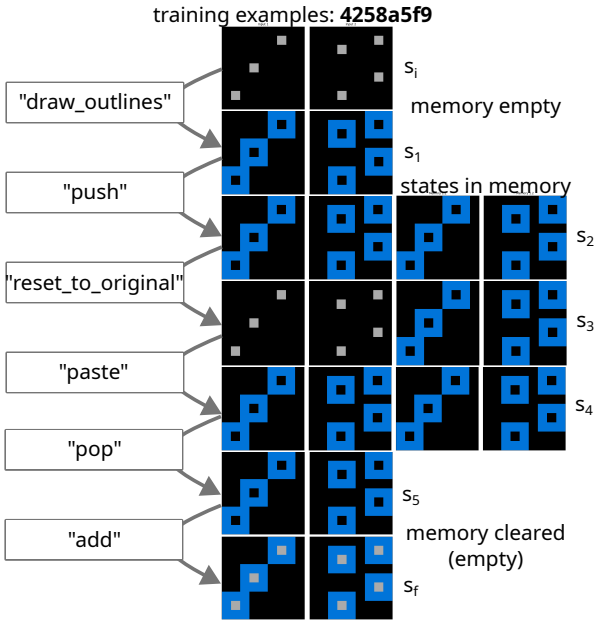

Figure 8: Example of usage of our memory actions in task **4258a5f9**, using 6 steps. OBS: Memory operations wouldn't be necessary for this task if action **"draw_outlines"** would return the outlines together with the original centers from state s_i, however it is not the case for all ARC tasks, making these operations necessary.

5 Combining Intuition and DSL

We propose a hybrid approach combining our abstract task representations with DSL-based sequential reasoning to address ARC's fundamental challenges. While deep learning models excel at pattern recognition, ARC's combinatorial nature makes pure memorization-based approaches infeasible. Our solution leverages two complementary components:

First, the grid autoencoder generates compact, abstract representations of both the overall task intuition and intermediate states. These representations enable the identification of semantically similar tasks, providing valuable guidance for solution strategies. However, the complexity of direct input-to-output transformations necessitates decomposition into simpler, sequential steps.

Second, the DSL framework provides this decomposition capability, allowing complex transformations to be expressed as sequences of primitive operations. We posit that a sufficiently rich DSL vocabulary can solve all public ARC tasks through multiple valid solution paths of varying lengths and compositions, mapping from initial state s_i to final state s_f. See Figure 9.

This approach suggests a new direction: creating a diverse dataset of human-generated solution sequences, where each task has multiple valid solutions. Unlike Large Language Models (LLMs), which cannot independently generate such varied solutions, humans can provide multiple solution strategies for each task. Combining with data augmentation techniques such as color permutation, this approach would enable a deep learning model to learn approaches while avoiding the challenges of extreme dimensionality and vast search spaces.

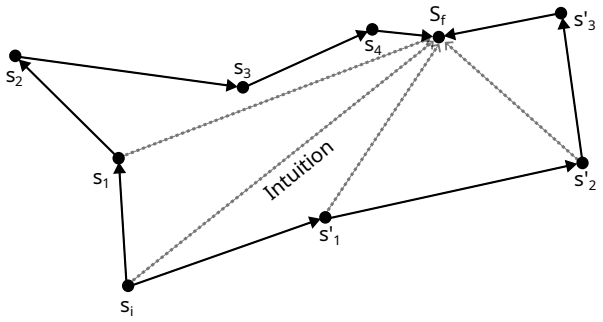

Figure 9: Example of a different possible paths for solving an ARC task. Two paths $\{s_i \to s_1 \to s_2 \to s_3 \to s_4 \to s_f\}$, and $\{s_i \to s'_1 \to s'_2 \to s'_3 \to s_f\}$. Dotted lines represent intuition vectors which can be computed at each step of the solution offering a compact observation space for future search programs.

6 Conclusion & Discussion

In this work, we have introduced a theoretical framework aimed at addressing the Abstraction and Reasoning Corpus (ARC) Challenge 2024. While our approach does not yet yield solutions for the ARC Prize 2024, we believe it establishes strong foundations for future advancements, particularly as we look towards 2025. Chollet envisions the solution to this challenge as a combination of discrete program search guided by deep learning intuition. Traditional approaches that attempt to learn direct mappings from input to output encounter the problem of data scarcity, as there are only 800 publicly available tasks within a search space of 10^{900} possible combinations. To address this, we propose utilizing deep learning's strength in memorization for a simpler task: reconstruction. By employing autoencoders, we present an abstract representation of ARC tasks, which is a recombination of latent representations of individual grids into what we call "intuition vectors." These compact representations carry semantic information about each task, and by applying UMAP projection on the 800 intuition vectors, we observed clusters of similar "intuition" that often corresponded to similar task objectives, such as pattern replication and object movement. This promising result suggests that such representations could inform systems about likely solutions for new, unseen tasks. While this represents

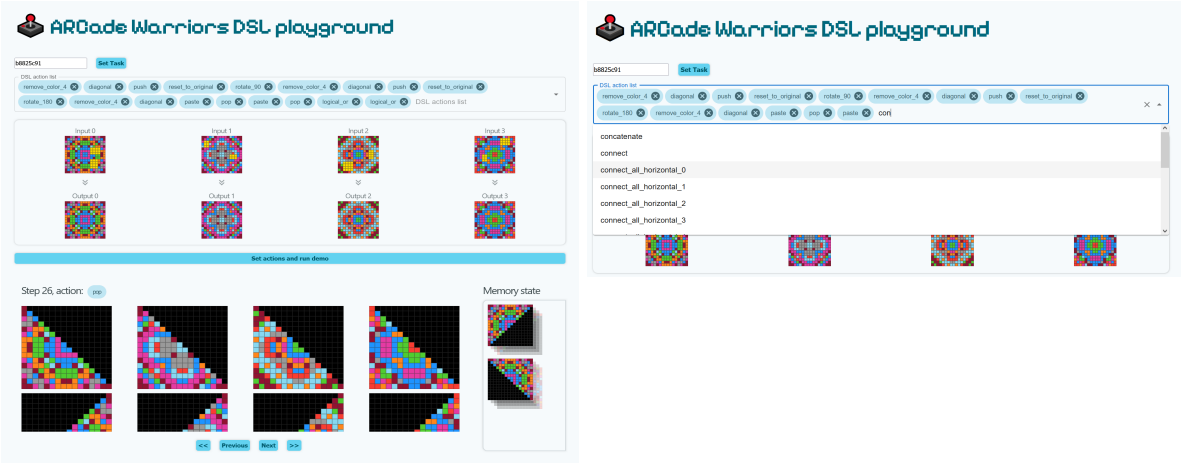

Figure 10: DSL implementation on our web interface. Users can produce novel DSL paths to solve publicly available tasks by concatenating non-parametric primitives. Solutions will be collected and distributed as a contribution or next year challenges.

significant progress in understanding ARC, the challenge lies not only in identifying similarities between known and new tasks but also in navigating the solution path precisely. Domain-Specific Languages (DSLs) have proven useful for this pathfinding problem by encapsulating human reasoning through discrete steps toward solutions. However, we believe the true potential of DSLs lies in collective human ingenuity. Studies on the ARC dataset have shown that individuals often describe tasks differently,

arriving at unique intermediate steps even when reaching the same final solution [LVLG24]. We propose that a dataset of multiple solutions for each task could unlock higher performance in future versions of this challenge. With this in mind, we developed a DSL featuring an addition of memory operations (push, paste, pop), which allows for the storage and retrieval of intermediate states. This enhancement enables any ARC task to be expressed as a sequence of parameter-free primitive functions, making our DSL both versatile and accessible.

To encourage collaboration and collective progress, we have made our DSL implementation and intuition models available to the public at `https://github.com/laurent-cheret/ARC.git`. We invite the research community to contribute new primitives and alternative solution paths, as we believe this collaborative approach is essential for advancing abstract reasoning in artificial intelligence. Additionally, we are developing a web-based interface that will serve as a platform for solving ARC tasks using our sequential approach, proposing new primitives, and reviewing existing ones. See Figure 10. We look forward to seeing this initiative inspire new insights and breakthroughs in the ARC challenge.

References

[BKG21] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders, 2021.

[Cho19] François Chollet. On the measure of intelligence, 2019.

[Hod23] Michael Hodel. Arc-dsl: Domain specific language for the abstraction and reasoning corpus. `https://github.com/michaelhodel/arc-dsl`, 2023.

[Hod24] Michael Hodel. Addressing the abstraction and reasoning corpus via procedural example generation, 2024.

[HVMR24] Rishi Hazra, Gabriele Venturato, Pedro Zuidberg Dos Martires, and Luc De Raedt. Can large language models reason? a characterization via 3-sat, 2024.

[JVLG21] Aysja Johnson, Wai Keen Vong, Brenden M. Lake, and Todd M. Gureckis. Fast and flexible: Human program induction in abstract reasoning tasks, 2021.

[LVLG24] 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.

[MHM18] L. McInnes, J. Healy, and J. Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. *ArXiv e-prints*, February 2018.

[VMSK23] Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. On the planning abilities of large language models : A critical investigation, 2023.

[WQR+24] Zhaofeng Wu, Linlu Qiu, Alexis Ross, Ekin Akyürek, Boyuan Chen, Bailin Wang, Najoung Kim, Jacob Andreas, and Yoon Kim. Reasoning or reciting? exploring the capabilities and limitations of language models through counterfactual tasks, 2024.