Vector Symbolic Algebras for the Abstraction and Reasoning Corpus

Isaac Joffe
Centre for Theoretical Neuroscience
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, ON
`ijoffe@uwaterloo.ca`

Chris Eliasmith
Centre for Theoretical Neuroscience
Department of Philosophy
Department of Systems Design Engineering
University of Waterloo
Waterloo, ON
`celiasmith@uwaterloo.ca`

Abstract

The Abstraction and Reasoning Corpus for Artificial General Intelligence (ARC-AGI) is a generative, few-shot fluid intelligence benchmark. Although humans effortlessly solve ARC-AGI, it remains extremely difficult for even the most advanced artificial intelligence systems. Inspired by methods for modelling human intelligence spanning neuroscience to psychology, we propose a cognitively plausible ARC-AGI solver. Our solver integrates System 1 intuitions with System 2 reasoning in an efficient and interpretable process using neurosymbolic methods based on Vector Symbolic Algebras (VSAs). Our solver works by object-centric program synthesis, leveraging VSAs to represent abstract objects, guide solution search, and enable sample-efficient neural learning. Preliminary results indicate success, with our solver scoring 10.8% on ARC-AGI-1-Train and 3.0% on ARC-AGI-1-Eval. Additionally, our solver performs well on simpler benchmarks, scoring 94.5% on Sort-of-ARC and 83.1% on 1D-ARC—the latter outperforming GPT-4 at a tiny fraction of the computational cost. Importantly, our approach is unique; we believe we are the first to apply VSAs to ARC-AGI and have developed the most cognitively plausible ARC-AGI solver yet. Our code is available at: `https://github.com/ijoffe/ARC-VSA-2025`.

1 Introduction

The Abstraction and Reasoning Corpus for Artificial General Intelligence (ARC-AGI; Chollet, 2019; Chollet et al., 2025b) challenges the tremendous progress deep learning has made in artificial intelligence (AI). ARC-AGI is a fluid intelligence benchmark comprising a collection of grid prediction tasks (see Figs. 1 and 2). The goal of the test-taker, human or AI, is simple: given a few pairs of input and output grids containing abstract symbols, determine the rules underlying the symbol transformations and use this understanding to predict the output grids corresponding to lone test input grids. ARC-AGI is easy for humans. Human performance is estimated to be 85.0% and all tasks were solved by at least one of two experts (Chollet et al., 2025a), indicating human intelligence can completely solve the benchmark. Conversely, ARC-AGI is hard for AI. Despite ample focus and investment over multiple $1,000,000 competitions (Chollet et al., 2024, 2025c), the benchmark remains unbeaten. ARC-AGI has proved resistant to old and new deep learning techniques, including the large language model (LLM)—OpenAI's original GPT-3 (Brown et al., 2020) scored 0.0% (Chollet et al., 2025a) notwithstanding claims of emergent reasoning (Wei et al., 2022). The chasm between human and AI performance on ARC-AGI suggests fundamental problems with leading approaches to AI.

Two such problems often identified in the literature are sample-efficient learning and explicit reasoning (Bengio et al., 2021; Greff et al., 2020; LeCun et al., 2015). Deep learning's success requires vast amounts of often-labelled high-quality training data, making it ineffective in the sample-few regime. Additionally, deep learning systems struggle with representing explicit rules, sometimes manifesting in poor out-of-distribution generalization. Vector Symbolic Algebras (VSAs), a neurosymbolic AI method introducing syntactic structure into high-dimensional distributed representations, hold promise to overcome these two limitations that ARC-AGI targets. VSAs specify high-dimensional

vectors to represent structured low-dimensional data, discrete or continuous. Thus, instead of learning useful embeddings from huge datasets, a VSA can be used to encode data with desired inductive biases. VSAs also define operators for the composition and systematic processing of data. Thus, instead of learning shortcut transformations that may generalize poorly to unseen inputs, a VSA, despite relying on distributed representations, can be used to express certain operations analytically. Additionally, VSAs support discrete search and neural learning, both of which are helpful for ARC-AGI. These strengths of VSAs, along with their ability to effectively model cognitive processes in a biologically plausible manner (Eliasmith and Anderson, 2003; Eliasmith, 2013), make them an excellent candidate for helping solve ARC-AGI.

We propose a novel, neurosymbolic, cognitively plausible ARC-AGI solver. Our solver uses VSAs, a general cognitive modelling framework, to bridge intuition and reason and capture the efficiency and interpretability of human intelligence. We believe we are the first to apply VSAs to ARC-AGI and, in doing so, have developed the most cognitively plausible ARC-AGI solver yet. The rest of this paper is organized as follows:

- Section 2 provides the background knowledge necessary to understand our work. We introduce the VSA used, discuss motivating history, and describe ARC-AGI further.
- Section 3 explains our approach. We detail and justify each part of our solver's design.
- Section 4 presents our results. We examine our solver's performance on ARC-AGI and related datasets.
- Section 5 analyzes our approach. We outline our solver's strengths and report its weaknesses.
- Section 6 summarizes our work.

2 Preliminaries

2.1 Vector Symbolic Algebras

Before explaining how our solver—which uses VSAs extensively—works, we describe VSAs further. VSAs, also known as hyperdimensional computing, are a family of algebras defining operations over a high-dimensional vector space. In this work, we use Holographic Reduced Representations (HRRs; Plate, 1991, 1995, 2003), one particular VSA.

2.1.1 Holographic Reduced Representations

HRRs define four operations: *similarity*, *binding*, *bundling*, and *inversion*. Following Furlong and Eliasmith (2022), we describe each.

Similarity, denoted by the $\cdot : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}$ operator, operates on two vectors, a and b, to produce a scalar, $s = a \cdot b$. The resultant value represents the semantic similarity between the vectors. Similarity is implemented as the vector dot product: $s = a \cdot b = \langle a, b \rangle = \|a\| \|b\| \cos \theta_{ab}$. Because we usually work with unit-length vectors (i.e., $\|a\| = \|b\| = 1$), the dot product is equivalent to cosine similarity and, thus, s falls between -1 and 1. Similarity is commutative. In high-dimensional vector spaces, the similarity between any two randomly-generated vectors is likely to be extremely small. Similarity can be used to *clean-up* noisy vectors by replacing the noisy vector with the one from a library of known vectors to which the noisy vector is most similar.

Bundling, denoted by the $+ : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}^N$ operator, combines two vectors, a and b, into one, $c = a + b$. The resultant vector is similar to both input vectors (i.e., $c \cdot a \gg 0$ and $c \cdot b \gg 0$); bundling gathers multiple vectors into a single representation, producing a superposition of those vectors. Bundling is implemented as element-wise vector addition. Bundling is commutative and associative.

Binding, denoted by the $\circledast : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}^N$ operator, combines two vectors, a and b, into one, $c = a \circledast b$. The resultant vector is dissimilar to both input vectors (i.e., $c \cdot a \simeq 0$ and $c \cdot b \simeq 0$); binding associates two vectors, producing a new vector unlike any existing vectors. Binding is implemented as circular convolution: $c = a \circledast b = \mathcal{F}^{-1} \{\mathcal{F}\{a\} \odot \mathcal{F}\{b\}\}$, where $\mathcal{F}$ is the discrete Fourier transform (DFT) and $\odot$ denotes Hadamard, or element-wise, multiplication. Binding is commutative and associative.

Inversion, denoted by the $^{-1} : \mathbb{R}^N \to \mathbb{R}^N$ operator, converts one vector, a, into another, a^{-1}. The resultant vector bound with the input vector approximately produces the binding identity vector (i.e., $a \circledast a^{-1} \simeq I$ where $a \circledast I = a$). Inversion can be used to *unbind* bound vectors: given a bound vector, $c = a \circledast b$, and one of its constituent vectors, a, the other can be extracted by binding with the inverse of the known constituent (i.e., $c \circledast a^{-1} = (a \circledast b) \circledast a^{-1} \simeq b$).

These four operations can be combined to encode and compute useful functions on complex data structures, such as lists (Choo, 2010) and graphs (Laube, 2024). For example, we can use a *slot-filler* representation to store multiple features of an item in a compressed but interpretable format. Given known slot vectors SIZE, SPECIES $\in \mathbb{R}^N$, we can
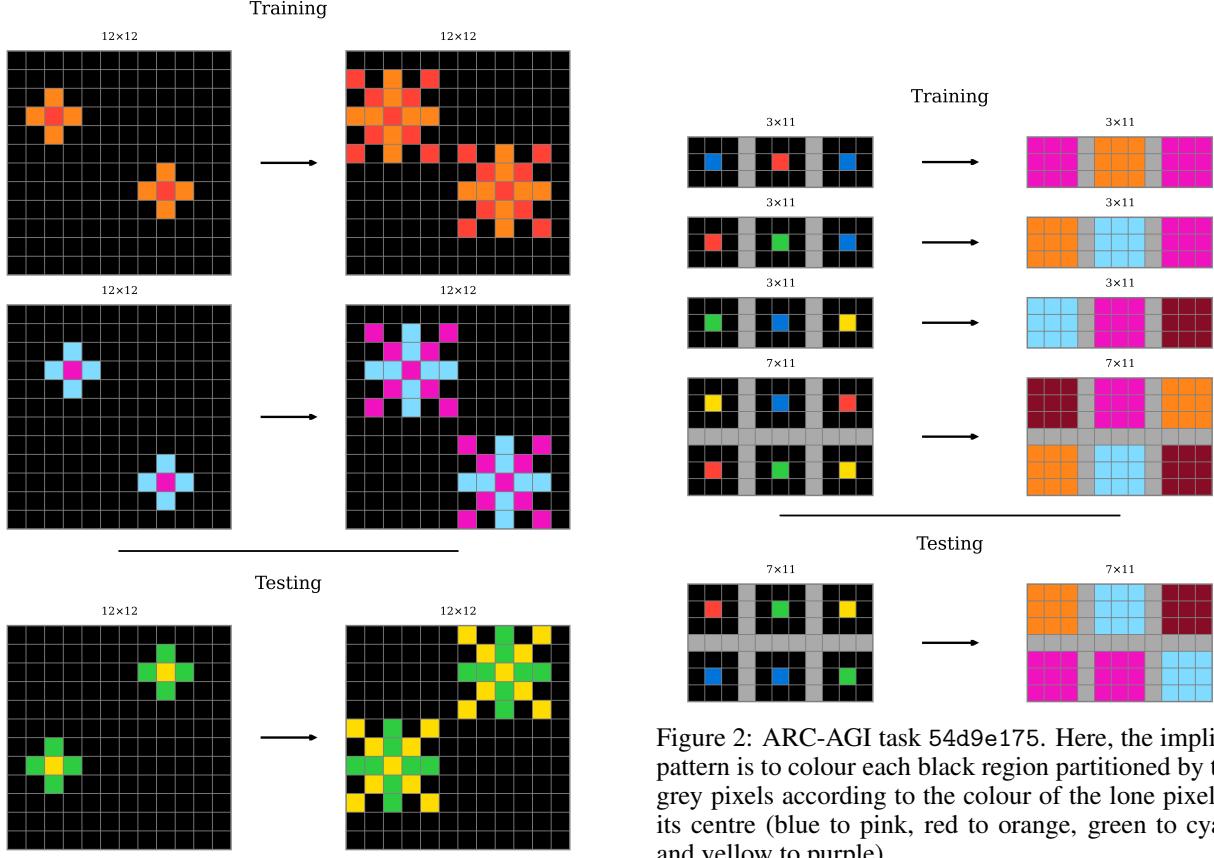
Figure 1: ARC-AGI task `0962bcdd`. Here, the implicit pattern is to transform each multi-coloured object into a larger object.

Figure 2: ARC-AGI task `54d9e175`. Here, the implicit pattern is to colour each black region partitioned by the grey pixels according to the colour of the lone pixel at its centre (blue to pink, red to orange, green to cyan, and yellow to purple).

encode a small dog as $\boldsymbol{a} = \texttt{SIZE} \circledast \texttt{SMALL} + \texttt{SPECIES} \circledast \texttt{DOG}$ and a large cat as $\boldsymbol{b} = \texttt{SIZE} \circledast \texttt{LARGE} + \texttt{SPECIES} \circledast \texttt{CAT}$, for instance. We can then query information about each object by unbinding: $\boldsymbol{a} \circledast \texttt{SIZE}^{-1} \simeq \texttt{SMALL}$ and $\boldsymbol{b} \circledast \texttt{SPECIES}^{-1} \simeq \texttt{CAT}$. The former query becomes

$$
\begin{aligned}
\boldsymbol{a} \circledast \texttt{SIZE}^{-1} &= \left(\texttt{SIZE} \circledast \texttt{SMALL} + \texttt{SPECIES} \circledast \texttt{DOG} \right) \circledast \texttt{SIZE}^{-1} \\
&= \texttt{SIZE} \circledast \texttt{SMALL} \circledast \texttt{SIZE}^{-1} + \texttt{SPECIES} \circledast \texttt{DOG} \circledast \texttt{SIZE}^{-1} \\
&\simeq \texttt{SMALL} + \texttt{SPECIES} \circledast \texttt{DOG} \circledast \texttt{SIZE}^{-1} \\
&\simeq \texttt{SMALL}
\end{aligned}
\tag{1}
$$

which works because $\texttt{SPECIES} \circledast \texttt{DOG} \circledast \texttt{SIZE}^{-1}$ behaves as noise that can be eliminated by clean-up.

Furthermore, we can encode a list of k items as $\boldsymbol{m} = \sum_i^k \texttt{ITEM}_i \circledast \texttt{ONE}^i$, where $\texttt{ONE}^i$ is the i-th index of the sequence (Choo, 2010). The slots, $\texttt{ONE}^i$, are defined according to a recursively-bound index vector, $\texttt{ONE}$, as

$$
\begin{aligned}
\texttt{TWO} &= \texttt{ONE} \circledast \texttt{ONE} = \texttt{ONE}^2 \\
\texttt{THREE} &= \texttt{TWO} \circledast \texttt{ONE} = \texttt{ONE} \circledast \texttt{ONE} \circledast \texttt{ONE} = \texttt{ONE}^3 \\
\texttt{FOUR} &= \ldots
\end{aligned}
\tag{2}
$$

where use of the exponentiation symbol is purely for notational convenience; the underlying operation is not repeated multiplication, but repeated binding. With this representation, the index of a known element in the list or the element at a known index in the list can be queried.

The Semantic Pointer Architecture (SPA; Eliasmith, 2013) is an architecture for modelling biological cognition using HRRs. The SPA has been used (Rasmussen, 2010; Rasmussen and Eliasmith, 2011) to solve the Raven's Progressive Matrices (RPMs; Raven, 1936) by which ARC-AGI was inspired. This RPM solver worked by generating VSA

representations of each cell in the grid based on perceptual outputs, unbinding each cell's representation from its subsequent cell's representation to obtain transformation vectors, averaging these transformation vectors to obtain a single rule vector, and applying this rule vector to the second-last cell to predict the final cell. ARC-AGI is a significantly harder problem because of its less constrained format (e.g., the requirement to generate solutions from scratch) and content (e.g., each task is distinct), but this past success suggests VSAs hold potential.

2.1.2 Spatial Semantic Pointers

Typical HRRs can represent and express a multitude of discrete, but not continuous, information structures and operations. Expanding on early suggestions (Plate, 1992), Spatial Semantic Pointers (SSPs; Komer et al., 2019; Komer, 2020) extend HRRs to continuous spaces. These representations can model (Dumont and Eliasmith, 2020) grid cell neurons observed in the brain (Doeller et al., 2010; Hafting et al., 2005) and can be used to solve challenging spatial reasoning tasks (Dumont et al., 2022, 2023; Dumont, 2025).

An SSP encoding defines a mapping $\phi : \mathbb{R}^D \to \mathbb{R}^N$ from a low, D-dimensional vector space, *feature space*, to a high, N-dimensional vector space, *SSP space*, based on a generalization of recursive binding into fractional binding. To understand this, consider a one-dimensional feature space. Before, the vector representing the n-th index of the sequence, $\texttt{ONE}^n$, was computed by binding $\texttt{ONE}$ with itself n times. However, we can also compute $\texttt{ONE}^n$ as

$$
\begin{aligned}
\texttt{ONE}^n &= \underbrace{\texttt{ONE} \circledast \ldots \circledast \texttt{ONE}}_{n \text{ times}} \\
&= \mathcal{F}^{-1} \left\{ \underbrace{\mathcal{F}\{\texttt{ONE}\} \odot \ldots \odot \mathcal{F}\{\texttt{ONE}\}}_{n \text{ times}} \right\} \\
&= \mathcal{F}^{-1} \left\{ \mathcal{F}\{\texttt{ONE}\}^n \right\}
\end{aligned}
\tag{3}
$$

where ^{n} represents element-wise exponentiation. This formulation allows natural generalization from $n \in \mathbb{Z}^+$ to $x \in \mathbb{R}$; instead of encoding discrete values by repeatedly binding the vector with itself, we can encode discrete or continuous values by directly exponentiating the DFT of the vector and taking the inverse DFT of this result. This formulation can be further generalized to represent multi-dimensional features, $\boldsymbol{x} \in \mathbb{R}^D$, by binding together the vectors representing each feature dimension. For example, consider the two-dimensional feature space $\mathbb{R}^2$. Given basis or axis vectors $\texttt{X}, \texttt{Y} \in \mathbb{R}^N$, an arbitrary point in the feature space $\boldsymbol{x} = (x, y)$ can be expressed in the SSP space as $\phi(\boldsymbol{x}) = \texttt{X}^x \circledast \texttt{Y}^y = \mathcal{F}^{-1} \{ \mathcal{F}\{\texttt{X}\}^x \odot \mathcal{F}\{\texttt{Y}\}^y \}$.

In the most general form, an SSP encoding is defined as

$$
\phi(\boldsymbol{x}) = \mathcal{F}^{-1} \left\{ e^{i\Theta \frac{\boldsymbol{x}}{l}} \right\}
\tag{4}
$$

where $\Theta \in \mathbb{R}^{D \times N}$ is the phase matrix of the SSP encoding and $l \in \mathbb{R}$ is a length-scale parameter. The phase matrix defines the mapping and ensures all SSPs generated are real-valued. Particular choices of Θ produce grid-cell-like representations (Dumont and Eliasmith, 2020).

SSPs can also be interpreted as probability representations over the feature space (Furlong and Eliasmith, 2022, 2023). This is because similarity induces a kernel function (Frady et al., 2022; Voelker, 2020): $\phi(\boldsymbol{x}_1) \cdot \phi(\boldsymbol{x}_2) = k(\phi(\boldsymbol{x}_1), \phi(\boldsymbol{x}_2))$. The behaviour of this kernel function, $k(\cdot, \cdot)$, can be modified by changing the distribution from which the phase matrix of the encoding is sampled. Other HRR operations also behave specially on SSPs. Binding is addition in the feature space: $\phi(\boldsymbol{x}_1) \circledast \phi(\boldsymbol{x}_2) = \phi(\boldsymbol{x}_1 + \boldsymbol{x}_2)$. Note that this makes shifting items easy and means the zero vector in the feature space (i.e., the origin) is represented as the identity vector in the SSP space (i.e., $\phi(\boldsymbol{0}) = \boldsymbol{I}$). This is because $\phi(\boldsymbol{x}) = \phi(\boldsymbol{x} + \boldsymbol{0}) = \phi(\boldsymbol{x}) \circledast \phi(\boldsymbol{0})$ and $\phi(\boldsymbol{x}) = \phi(\boldsymbol{x}) \circledast \boldsymbol{I}$. Inversion is negation in the feature space: $\phi(\boldsymbol{x})^{-1} = \phi(-\boldsymbol{x})$. This is because $\boldsymbol{I} = \phi(\boldsymbol{0}) = \phi(\boldsymbol{x} + (-\boldsymbol{x})) = \phi(\boldsymbol{x}) \circledast \phi(-\boldsymbol{x})$ and $\boldsymbol{I} = \phi(\boldsymbol{x}) \circledast \phi(\boldsymbol{x})^{-1}$.

SSPs can be combined with other vectors to represent complex scenes. For example, we can encode k items at certain positions in a scene as $\boldsymbol{a} = \sum_i^k \texttt{ITEM}_i \circledast \phi(\boldsymbol{x}_i)$. As in the discrete list example, the position of a known item in the scene or the item at a known position in the scene can be queried.

2.2 Dual Process Cognition

We often explain the operation of our solver with metaphors to dual process theory (Wason and Evans, 1974), supporting cognitive plausibility. To elucidate these comparisons, we introduce dual process theory and its applications to both psychology and AI.

2.2.1 System 1 and System 2

In psychology, the mind is often understood as being composed of two complementary systems, System 1 and System 2 (Kahneman, 2011). System 1 performs fast, automatic tasks, and can be thought of as a fictitious agent modelled as a large associative memory tuned by evolution and experience. System 1 operates constantly and involuntarily, appraising the environment to invoke emotions and produce impressions that generate a coherent world model. The operation of System 1 is characterized by heuristics, accurate but imperfect intuitions that intelligently guide behaviour. System 2 performs slow, intentional tasks, and can be thought of as another fictitious agent aligning with our conscious experience. System 2 operates sparingly and intentionally, solving complex problems only when needed because mobilizing the required cognitive resources is expensive. The operation of System 2 is characterized by calculated reasoning for attention-intensive tasks. System 1 and System 2 each play a role in intelligence because they solve different problems, so it is often argued that a complete account of intelligence must model both (Bengio et al., 2021).

2.2.2 Symbolic and Connectionist Intelligence

Early approaches to AI (e.g., Lindsay et al., 1993; Weizenbaum, 1966), inspired by formal reasoning, sought to emulate symbolic cognition (Newell and Simon, 1976; Newell, 1980). In this paradigm, intelligence arises from strictly following explicit, hand-crafted rules. The resulting so-called expert systems excel at conveying narrow domain knowledge and performing logical reasoning, but suffer from important limitations. First, explicit symbolic rules are brittle, robust to neither changes in the environment nor noisy data in raw, perceptual form (d'Avila Garcez and Lamb, 2023; Harnad, 1990; Kautz, 2022). Second, discrete search is not scalable, quickly becoming intractable due to inevitable combinatorial explosion (Kautz, 2022). Third, learning is difficult, limiting generalizability and constraining systems to narrow scopes of expertise (Garnelo and Shanahan, 2019; Nawaz et al., 2025). Fourth, biological implementation of the symbolic paradigm remains unclear.

Recent successes in AI (e.g., Krizhevsky et al., 2012; Radford et al., 2019; Silver et al., 2016), inspired by the observation of countless neurons activating in concert in the brain (McCulloch and Pitts, 1943; Rosenblatt, 1958), have been dominated by the connectionist view (Rumelhart et al., 1986b). In this paradigm, intelligence arises from operations computed as functions of high-dimensional feature representations in large-scale neural networks (NNs; LeCun et al., 2015; Schmidhuber, 2015). The resulting neural and statistical machines are extremely powerful at extracting patterns from data, but suffer from important limitations of their own (Fodor and Pylyshyn, 1988). First, connectionist models scale to a fault, performing better with increased model and dataset size but requiring large models and datasets to succeed (d'Avila Garcez and Lamb, 2023; Garnelo and Shanahan, 2019). Second, connectionist models learn to a fault, sometimes memorizing their training data and often generalizing poorly outside of their training distribution (d'Avila Garcez and Lamb, 2023; Garnelo and Shanahan, 2019; Goodfellow et al., 2015; Marcus, 2020; Szegedy et al., 2014). Third, learned high-dimensional representations and operations are opaque, rendering many connectionist models unexplainable black boxes (d'Avila Garcez and Lamb, 2023; Garnelo and Shanahan, 2019; Nawaz et al., 2025; Zhang and Sheng, 2024). Fourth, biological implementation of the connectionist paradigm also remains unclear (Crick, 1989; Rumelhart et al., 1986a).

Connectionist models excel at System 1 tasks, such as recognizing images, transcribing speech, and generating text. Conversely, symbolic models excel at System 2 tasks, such as logical inference and trial-and-error search. We believe neurosymbolic models integrating elements of each paradigm are necessary to completely model intelligence.

VSAs provide one type of neurosymbolic approach. VSAs bridge symbolic and connectionist AI, combining the strengths of each to mitigate the weaknesses of each. Symbol-like representations afford the benefits of explicit, hand-crafted rules while overcoming the limitations of pure symbolic models. Using distributed representations provides robustness to noise, facilitates similarity metrics to guide search, and enables continuous learning. Neural-like representations afford the benefits of neural learning in NNs while overcoming the limitations of pure connectionist models. Using structured representations with useful inductive biases and features pre-encoded improves sample-efficiency in learning, enables generalizable algebraic models, and permits direct interpretation of intermediate representations. Additionally, VSAs are biologically and cognitively plausible and can be implemented in spiking NNs (Eliasmith and Anderson, 2003; Eliasmith, 2013).

2.3 Abstraction and Reasoning Corpus

Before explaining how our solver works, we provide a more detailed description of ARC-AGI. Here, we discuss its theoretical foundations, its exact format, how others have approached it, and how we approach it.

2.3.1 Background and Definition

AI was originally defined as "the science of making machines do things that would require intelligence if done by [people]" (Minsky, 1968). This skill-based view of intelligence underpins many modern AI systems adept at one particular task, but only that one task. The skills AI systems can perform—synthetic art generation and natural language translation, for instance—have become increasingly advanced, but the underlying paradigm remains the same: models fit their many parameters to copious amounts of data during a training phase, and statically apply their learned skill to unseen data during an inference phase. In contrast, humans are simultaneously adept at many tasks. A skilled human may be able to both create art and translate between languages, and unskilled humans can learn these new skills. In this sense, human intelligence is better characterized by its adaptability, fluidity, and generality. Not only can we humans excel at particular well-known tasks, we can rapidly learn to perform new tasks in response to new situations. Human intelligence is dynamic, without disjoint training and inference phases. In ARC-AGI's initial introduction, Chollet defined intelligence as skill-acquisition efficiency: the ability of a system to quickly develop novel skills to solve novel problems (Chollet, 2019). In this view, crystalline skill at any number of tasks is not intelligence; instead, intelligence is the process by which those skills are learned. ARC-AGI aligns with this paradigm by testing the ability of a system to learn arbitrary transforms on-the-fly from few examples.

ARC-AGI is a collection of abstract reasoning *tasks* (see Figs. 1 and 2). The original dataset, ARC-AGI-1, consists of 400 public training tasks, 400 public evaluation tasks, and 200 private evaluation tasks. The newer version, ARC-AGI-2, consists of 1000 public training tasks, 120 public evaluation tasks, and 120 private evaluation tasks. The performance of a solver is usually assessed on each group separately because the evaluation splits are more difficult. The private splits are known only to the ARC-AGI team and are used to test the true generalizability of solvers. Each task has a *training* component and a *testing* component. The training component is a set, $\mathcal{D}$, of a few *demonstrations*, or demonstration pairs. The number of demonstrations is a feature of the task, but is usually about three (in ARC-AGI-1-Train, $2 \leq |\mathcal{D}| \leq 10$ with median 3, mean 3.26, and standard deviation 0.96). Each demonstration is composed of two *grids*, one *input* grid and one *output* grid. Each grid, $\mathbf{G} \in \mathbb{G}$, contains r rows and c columns of *pixels*. Grids must be rectangular; they are often square, but r need not equal c. Grids range in size (in ARC-AGI-1-Train, $r, c \in \{1, \dots, 30\}$ with medians 10 and 10, means 9.64 and 10.03, and standard deviations 6.01 and 6.16); the sizes of the input and output grids in a demonstration often match, but a grid size change may be part of the task. Each pixel, $P_{ij} \in \mathbb{P}$, is one of ten discrete symbols provided as an integer but usually displayed as a colour for human convenience. The exact colours used for display are arbitrary; no task relies on facts about colours, such as that orange is a mix of red and yellow. To summarize, $\mathcal{D} = \{(\mathbf{G}_{I_d}, \mathbf{G}_{O_d}) \mid \mathbf{G}_{*_*} \in \mathbb{G}, d = 1 \dots |\mathcal{D}|\}$, where $\mathbb{G} = \mathbb{P}^{r_{**} \times c_{**}}$ and $\mathbb{P} = \{0, \dots, 9\}$. The input grid of each demonstration can be mapped onto its corresponding output grid via some *transformation*. This transformation, $\mathcal{T}_d$, describes how to produce this specific output grid, $\mathbf{G}_{O_d}$, from the corresponding input grid, $\mathbf{G}_{I_d}$. The transformations underlying each demonstration in a task share structure and can be generalized into a simple common *program*. This program, $\mathcal{P}$, describes how to produce the corresponding output grid for any valid input grid. To summarize, $\mathbf{G}_{I_d} \xrightarrow{\mathcal{T}_d} \mathbf{G}_{O_d}$ for each $d \in \{1, \dots, |\mathcal{D}|\}$ and $\mathbf{G}_{I_d} \xrightarrow{\mathcal{P}} \mathbf{G}_{O_d} \ \forall d \in \{1, \dots, |\mathcal{D}|\}$. The testing component is also a set, $\mathcal{Q}$, of a few *queries*. The number of queries is also a feature of the task, but is usually just one (in ARC-AGI-1-Train, $1 \leq |\mathcal{Q}| \leq 3$ with median 1, mean 1.04, and standard deviation 0.22). A query is a lone test input grid without a provided output grid. The goal of the solver is to correctly answer the queries; to solve a task correctly, the solver must predict exactly the correct output grid, both its size and contents, for all queries. The performance of a solver is defined as the proportion of some subset of tasks solved correctly.

What makes ARC-AGI so difficult is also what makes it compelling. There are several important differences between ARC-AGI and other benchmarks. First, the problem is generative. Instead of choosing the correct output grid from a set of candidates as in classification tasks, such as the RPMs, solvers must generate the output grid from scratch. Second, the reasoning involved is, as the name suggests, abstract. Implied objects are not grounded (i.e., they may have no real-world correlates), implied actions invoke high-level ideas (e.g., motion until contact), and implied rules invoke relative and fuzzy concepts (e.g., the largest among a set or the horizontalness of a shape). As such, solvers must represent each grid in terms of its abstract qualities and construct a program insensitive to inconsequential details of the particular demonstrations. Third, no two tasks are alike. No solution program will work for any pair of tasks, and the possible concepts, transforms, and their compositions are wildly unconstrained.

Despite these complexities, ARC-AGI is based only on four core knowledge priors: objectness, goal-directedness, numbers, and geometry. Objectness requires perceiving grids as collections of cohesive, persistent, interacting objects. Goal-directedness requires anthropomorphizing objects into agents acting to achieve goals. Numbers requires understanding basic arithmetic, such as counting, addition, and subtraction, and abstract mathematical notions, such as sorting, smallest, and largest. Geometry requires understanding position, symmetry, translation, rotation, scaling, and the like. A solver with a thorough understanding of only these four concepts can solve ARC-AGI; specialized

knowledge, such as language or external world facts, are unnecessary. Humans innately have some of these priors from evolution, and acquire the others through experience interacting with the modern world.

2.3.2 Related Work

Many ARC-AGI solvers can be divided into two broad classes: *transductive* solvers and *inductive* solvers (Li et al., 2025; Chollet et al., 2025a). Transductive solvers (e.g., Akyürek et al., 2025; Bober-Irizar and Banerjee, 2024; Cole and Osman, 2025; Fletcher-Hill, 2024; Franzen et al., 2025; Puget, 2024; Wang et al., 2025) directly answer the queries without explicit intermediate reasoning. These approaches use an NN, usually an LLM, to implicitly understand the demonstration transformations and apply the query transformations all at once, without generating any representation of the patterns underlying the transformations. Inductive solvers (e.g., Ferré, 2024; Hocquette and Cropper, 2025; Lim et al., 2024; Neumann and Pintér, 2023; Rocha et al., 2025; Wind, 2020; Witt et al., 2025; Xu et al., 2023) generate explicit rules mapping input grids to output grids and apply these rules to the queries. These approaches use various search techniques, but invariably cast ARC-AGI as a program synthesis problem. Both approaches in isolation are problematic; neurosymbolic approaches (e.g., Banburski et al., 2020; Bednarek and Krawiec, 2024; Macfarlane and Bonnet, 2025; Ouellette, 2024; Thoms et al., 2023; Wang et al., 2024) integrating transduction and induction, such as deep-learning-guided program synthesis, have been suggested as the ideal solution (Chollet et al., 2025a).

Transductive solvers are dominated by LLMs and other transformer-based (Vaswani et al., 2017) models. But, despite their impressive abilities, LLMs suffer from several critical weaknesses that prove problematic for solving ARC-AGI. First, LLMs struggle with numeracy. Stemming from fundamental limitations of the transformer architecture (Ouellette et al., 2024; Yehudai et al., 2024), LLMs struggle to perform out-of-distribution arithmetic (Yuan et al., 2023) or, infamously, to even count the number of occurrences of a letter in a sequence (Fu et al., 2024). Second, LLMs struggle with following rules. As such, LLMs cannot play many games effectively (de Carvalho et al., 2025), and even their 'reasoning' variants cannot follow steps well enough to consistently solve complex problems (Shojaee et al., 2025). Third, LLMs struggle with out-of-distribution generalization. LLMs lack robustness and fail to exhibit even simple forms of generalization: they cannot generalize facts into their reversed form (Berglund et al., 2024), are biased by content effects (Lampinen et al., 2024), and brittly collapse on slight variations of known tasks (Wu et al., 2024; McCoy et al., 2024). Fourth, LLMs struggle with important capabilities of human intelligence. LLMs cannot learn fast, generalize broadly, or explicitly understand their actions (Nam and McClelland, 2024). Language is useful to express reasoning, but is not required to perform reasoning (Fedorenko et al., 2024). These failures of LLMs, along with their restriction to the text domain, suggest deficiencies in ARC-AGI's required core knowledge priors and intrinsic issues with performing the systematic reasoning required to solve ARC-AGI. LLMs, when prompted directly or with chain-of-thought, and vision language models in isolation have weak performance (Galanti and Baron, 2024; Lee et al., 2025; Mitchell et al., 2023; Singh et al., 2023).

Inductive solvers perform search on either a general-purpose language (GPL) or a domain-specific language (DSL). GPLs, such as Python, are expressive and can be used to solve all ARC-AGI tasks, providing scalability. But, GPLs lack the inductive biases necessary to express solutions to ARC-AGI simply, making search difficult. Conversely, DSLs, requiring hand-crafted primitives, allow humans to introduce problem-relevant inductive biases, making search easier. But, DSLs are not expressive enough to solve all possible ARC-AGI-like tasks, limiting scalability. DSLs capable of solving all existing ARC-AGI tasks (Hodel, 2023, 2024) are massive and still may be incapable of solving new tasks. Additionally, without proper guidance through the search space, search processes are unintelligent.

Unfortunately, little is known about how exactly humans solve ARC-AGI so effectively. But, some facts are clear from studies on humans. First, humans reason in terms of objects. This is true in general, and action traces of humans solving ARC-AGI tasks indeed show pixel groups are operated on together (Johnson et al., 2021). Second, humans dedicate time to explicit hypothesis generation. A non-negligible period of initial inaction is consistently observed (Johnson et al., 2021). Third, humans can express their task solutions programmatically. Transformations can be communicated in terms of natural language rules (Acquaviva et al., 2022); whether the solution discovery process itself is program synthesis remains unknown. Fourth, humans conceptualize features and operations using high-level abstractions. For example, humans describe their solutions in terms of objects with "tails" (Johnson et al., 2021), objects as "flowers" (LeGris et al., 2025), and objects "bumping into" each other (Acquaviva et al., 2022); whether such abstractions are necessary for the solving process remains unknown. Fifth, humans make mistakes different from those made by artificial solvers. Human failures usually show partial correctness indicating some understanding (LeGris et al., 2024, 2025), but the failures of artificial solvers indicate little understanding (Opiełka et al., 2024). We take these facts to suggest humans solve ARC-AGI by means of object-centric program synthesis.

Importantly, transductive and inductive solvers are each doing something entirely unlike what humans are doing; humans are not solving ARC-AGI purely by means of transduction or induction. Transductive solvers, based on connectionist networks, model System 1; inductive solvers, based on symbolic engines, model System 2. But, neither

is enough alone: humans are able to solve ARC-AGI so effectively because of contributions from both System 1 and System 2. Humans are guided by System 1 intuitions, immediate instincts arising from common sense and world knowledge, but also generate and test explicit hypotheses with System 2 reasoning, careful consideration of a limited number of ideas. We believe a neurosymbolic approach combining the strengths of symbolic and connectionist AI is thus needed.

3 Methods

Fundamentally, we aim to develop a cognitively plausible ARC-AGI solver. The goal of ARC-AGI is to spur progress towards artificial general intelligence (AGI), so taking inspiration from human intelligence, the only known general intelligence, is reasonable. Additionally, human intelligence remains the best ARC-AGI solver by a wide margin, suggesting important insights are missing from artificial solvers.

3.1 Solution Structure

We believe humans solve ARC-AGI via object-centric program synthesis. As such, our solver generates solutions to ARC-AGI tasks based on objects and programs.

3.1.1 Objects

Humans reason about ARC-AGI with *objects* rather than with individual pixels or whole grids. Fundamentally, an object is a group of pixels transformed cohesively. Because the transformations are different in every task, what makes a group of pixels constitute an object is task-dependent; the same group of pixels may form an object in one task, but not in another. Thus, for each task, an object definition must be discovered alongside the transformations as part of the solution generation process. Following this view, our solver represents each grid, input and output, as the set of its constituent objects and each transformation, training and testing, as a series of one-to-one operations on those objects.

Object representation determines both what tasks can be solved and how those tasks can be solved. Certain object properties may be required to solve some tasks, but not others. For example, an object's colour is relevant in ARC-AGI tasks 54d9e175 and a61f2674 (see Figs. 2 and 4), but irrelevant in ARC-AGI tasks 0962bcdd and a61ba2ce (see Figs. 1 and 3). Additionally, the way information is represented makes some transformations easier to implement, but others harder. For example, representing position in Cartesian coordinates makes performing translations simple but rotations complicated, and vice versa for polar coordinates.

We adopt an object representation scheme with three features: *colour*, *centre*, and *shape*. An object's colour is its discrete symbol in the grid. Colour is represented as one of ten vectors; as such, objects may only be one colour. An object's centre is the midpoint of the furthest extent of all its pixels in all directions. Centre is represented as the SSP encoding the two-dimensional coordinates of this midpoint relative to the grid's centre, with a blurring effect facilitating partial similarity. An object's shape is its pixel pattern. Shape is represented as the normalized bundle of the SSPs encoding each pixel's location relative to the object's centre. Object representations can be visualized using standard SPA and SSP methods (see Fig. 5). We hypothesize that higher-level object properties, such as symmetry and exact pixel count, initially go unnoticed by humans and are only encoded after their relevance becomes apparent. Although necessary to rigorously solve some tasks, these properties are not explicitly considered by our solver.

3.1.2 Programs

Humans solve ARC-AGI with *programs*. In this view, solving an ARC-AGI task means synthesizing a general program to generate the correct output grid for any valid input grid. Crucially, the search process underlying program discovery is both constrained and intentional. To solve an ARC-AGI task, humans iteratively explore multiple solution hypotheses—re-interpreting the contents of the grids and conceptualizing new transformations based on different abstractions—until they find a satisfactory algorithm to solve all demonstrations. However, the extent of this exploration is narrow; because System 2 cognitive resources are limited, humans consciously examine only a few solution hypotheses. System 1 intuitions guide which paths are explored; humans quickly dismiss nonsensical ideas and only seriously investigate ideas that conceptually fit the task's content. Following this view, our solver iteratively constructs and tests hypotheses represented as solution programs.

Program representation also determines both what tasks can be solved and how those tasks can be solved. Our solver synthesizes programs within a DSL. The primitives in this DSL were carefully designed to capture human inductive biases commonly appearing in ARC-AGI while remaining as general-purpose as possible. For example, our DSL contains a basic IDENTITY operation to exactly reproduce an input object as an output object, elementary RECOLOUR,
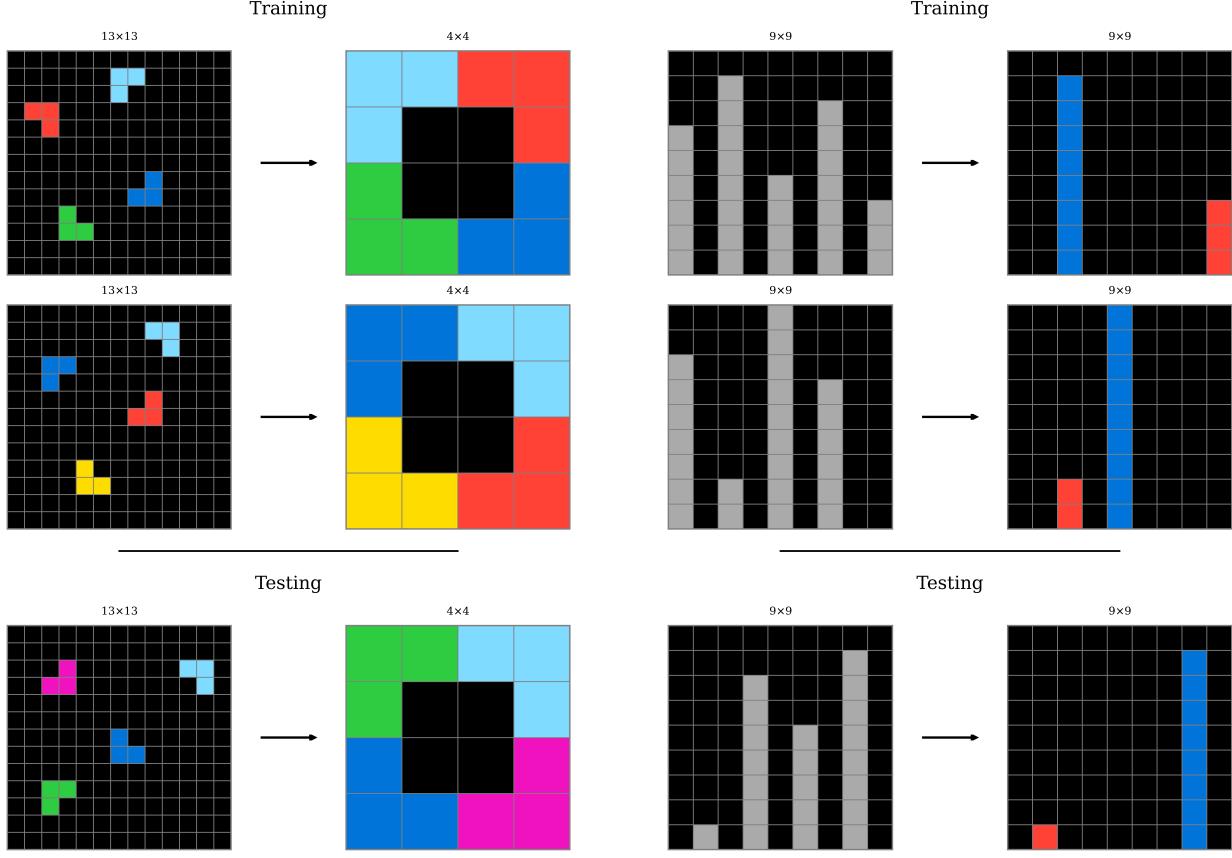
Figure 3: ARC-AGI task `a61ba2ce`. Here, the implicit pattern is to rearrange the four blocks into a hollow square on a small grid.

Figure 4: ARC-AGI task `a61f2674`. Here, the implicit pattern is to recolour the shortest stripe red and the tallest stripe blue.

RECENTRE, and RESHAPE operations to change each object property, an open-ended GENERATE operation to create an arbitrary output object, and more (see Table 1).

More formally, our solver produces a solution to each task as a *program* (see Fig. 6 for an example). This program can be instantiated to generate the correct transformation for each demonstration and, ideally, unseen valid input grids. We define a solution program as a set of *rules*. Each rule describes a different type of object-to-object mapping as an if-then statement expressing an *action* to be taken based on some *condition*. Conditions are logical compositions of *criteria*. Criteria reflect some state or feature of a particular object *property*. For example, the condition for a rule may be "COLOUR *is not* grey $\wedge$ SHAPE *is* one pixel", a conjunction of the criteria "COLOUR *is not* grey" and "SHAPE *is* one pixel" based on the COLOUR and SHAPE properties of an object. Conditions may be vacuous, causing the associated action to apply to all objects. Actions are particular *operations*, taken from our DSL, to apply to each object satisfying the rule's condition. Operations convert one input object into one output object in some structured way. Open-ended operations require *parameters*. Parameters determine how each instantiation of the operation should behave. For example, the action for a particular rule may be "GENERATE(COLOUR : orange; CENTRE : origin; SHAPE : 3×3 square)", representing the application of the GENERATE operation with the COLOUR parameter as orange, CENTRE parameter as the origin, and SHAPE parameter as a 3×3 square.

3.2 Solution Synthesis

Our solver models human cognition throughout all stages of the solving process. For example, humans often start solving ARC-AGI tasks by sizing the output grids to be generated (Johnson et al., 2021; LeGris et al., 2024). As such, our solver begins by generating a hypothesis about how the grid size changes in the task (see Table 2). If the sizes of the input and output grid are the same in each demonstration, then the size of each test output grid is predicted to the be the same as the corresponding query; otherwise, if the sizes of all output grids across all demonstrations are the
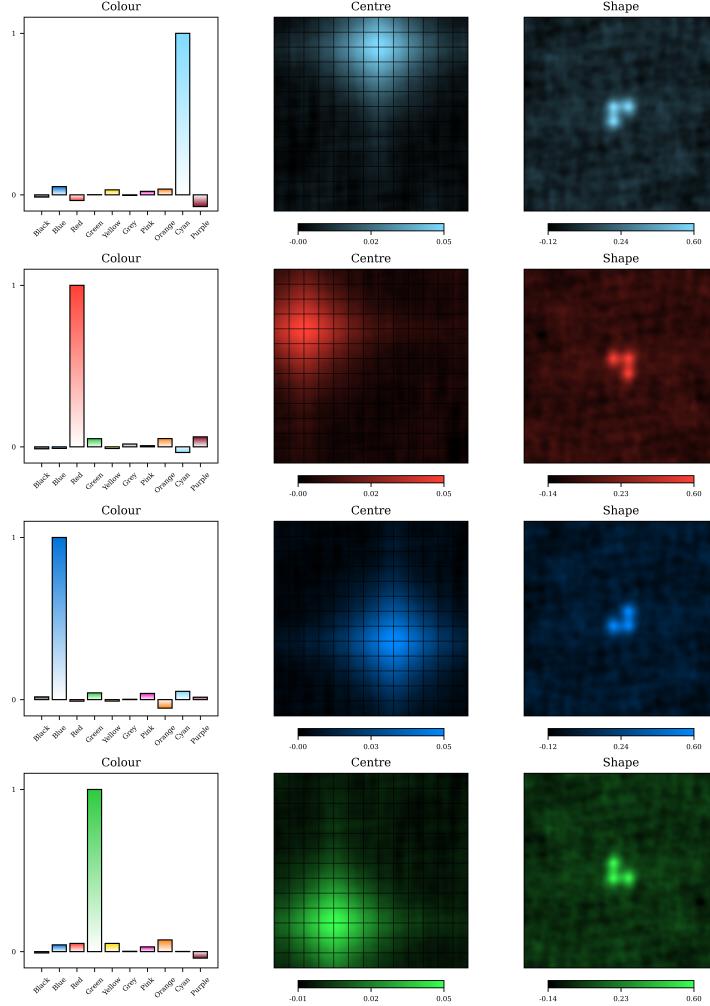
Figure 5: Visualization of the object representations for the input grid in the first demonstration of ARC-AGI task `a61ba2ce` (see Fig. 3). The colour subfigure (left) shows the similarity of the object's colour representation to each of the ten possible colour vectors. The centre subfigure (middle) shows the similarity of the object's centre representation to the SSP encoding each location in the grid. The shape subfigure (right) shows the similarity of the object's shape representation to the SSP encoding each possible location in two-dimensional space. These representations use $N = 1024$-dimensional vectors.
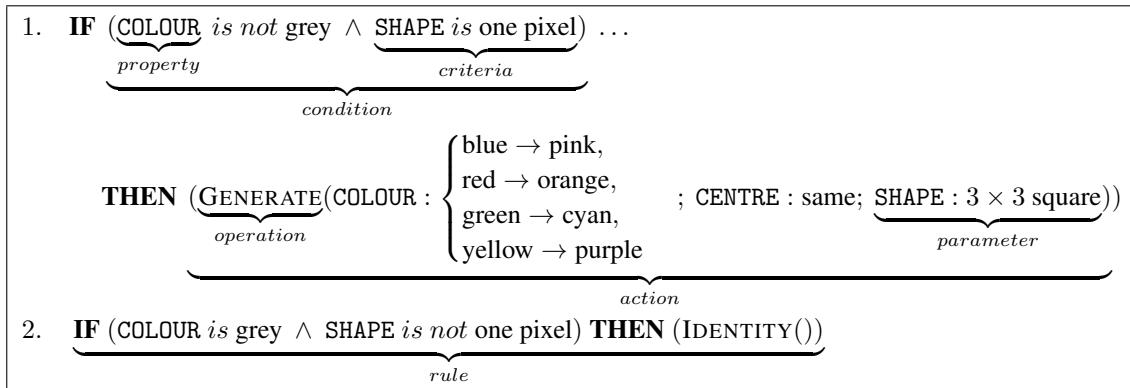

Figure 6: One possible solution program for ARC-AGI task `54d9e175` (see Fig. 2).

Table 1: Our solver's DSL.

Operation	Parameter(s)	Explanation
IDENTITY	—	Keeps the object as is.
EXTRACT	—	Shrinks the grid around the object.
RECOLOUR	`COLOUR`	Changes the object's colour to `COLOUR`.
RECENTRE	`CENTRE`	Changes the object's centre to `CENTRE`.
RESHAPE	`SHAPE`	Changes the object's shape to `SHAPE`.
MOVE	`AMOUNT`	Shifts the object's centre by `AMOUNT`.
GRAVITY	`DIRECTION`	Shifts the object's centre in `DIRECTION` until contact with another object or grid boundaries.
GROW	`DIRECTION`	Changes the object's centre and shape by stretching the object in `DIRECTION` until contact with another object or grid boundaries.
FILL	—	Fills in the object's shape.
HOLLOW	—	Hollows out the object's shape.
GENERATE	`COLOUR, CENTRE, SHAPE`	Creates an object with colour `COLOUR` and shape `SHAPE` at centre `CENTRE`.

same, then the size of each test output grid is predicted to be the same as this constant size; otherwise, the output grid sizes are determined by the actions in the solution program.

We believe humans solve ARC-AGI using multiple types of logical reasoning. As such, we divide the rest of our solver's solution generation process into three stages, each completed by one mode of reasoning: *demonstration abduction*, *rule induction*, and *answer deduction*. First, demonstration abduction generates an object definition and a set of simple steps to explain each demonstration (i.e., the objects and actions to produce each training output grid from its corresponding input grid). Simply understanding what is happening in each demonstration is necessary for further reasoning; abductive reasoning can find the most probable, usually the simplest, causal explanation for these observations. Second, rule induction generates broadly-applicable rules expressing when and how actions are applied, accounting for all observations. Finding the commonalities and distinctions underlying all observed action applications is necessary for true understanding; inductive reasoning can generalize these specific observations into rules. Third, answer deduction applies the rules generated to each query to produce a corresponding output grid. Whatever understanding of the demonstrations has been developed must be applied to the novel test input grids; deductive reasoning can draw logically sound inferences from these well-defined premises. Together, these three steps bridge multiple reasoning modes to model the entire ARC-AGI solving process.

3.2.1 Demonstration Abduction

First, our solver abduces the objects and actions in the demonstrations. Any solver must conceptualize what it observes in the demonstrations before it can generalize these observations into rules. This requires determining, in parallel, what exactly constitutes an object in this particular task and what specific actions on the input objects can produce the correct set of output objects. Both the object hypothesis and action hypotheses should be as simple as possible.

Determining both an object hypothesis and action hypotheses creates a causality paradox: an object is defined as a group of pixels transformed cohesively, but actions are ill-defined without objects to transform. We solve this paradox by searching for action hypotheses based on evolving object proposals. Our solver applies an initial object hypothesis

Table 2: Our solver's grid size change hypotheses.

Hypothesis	Description
IDENTITY	Each output grid is the same size as its corresponding input grid.
CONSTANT	All output grids are of the same constant size.
FUNCTION	The size of the output grid is some function of the size of the input grid or its contents.

and abduces actions for these objects; only when this object hypothesis fails or yields overly complex actions are other object hypotheses considered.

We believe humans do the same. When first seeing a new ARC-AGI grid, System 1 produces initial ideas of what the objects are. This is part of the objectness core knowledge prior; humans have strong inductive biases, innate and learned, about what objects look like. These initial object proposals are often correct, but, when they fail, humans mobilize System 2 to carefully consider other object hypotheses. This top-down approach works because the space of possible object definitions is smaller than the space of possible actions. A bottom-up approach constructing objects out of individual pixels transformed cohesively, in addition to being cognitively implausible, fails because cohesive object-level actions are often not cohesive pixel-level actions. For example, a rotation transformation is simple at the object level but ill-defined at the pixel level.

Developing a sufficient set of action hypotheses requires explaining the presence of each output object in each demonstration by the application of some action to one of the corresponding input objects. Difficulty arises because usually each output object can be explained by multiple actions applied to multiple input objects. This is because our DSL has operations with inherently overlapping functionality as well as a GENERATE operation that can represent arbitrary transformations from any input object.

There are multiple valid ways to solve each ARC-AGI task. Consider ARC-AGI task `a61ba2ce` (see Fig. 3); solving this task can be conceptualized as moving each object into its corresponding corner on a smaller grid, colouring each corner of a smaller grid according to the colour of the corresponding object, and so on. For this task, our solver's approach resembles the first description. In this view, the actions needed to complete each transformation, once the output grid is resized to 4×4, are to move the upper-left-corner-shaped object to the upper left corner of the grid, move the upper-right-corner-shaped object to the upper right corner of the grid, and so on. However, even this conceptualization can be implemented as a program in multiple ways. Our DSL has both a RECENTRE operation for absolute motion and a MOVE operation for relative motion; each operation proves useful for certain tasks, but this task calls for absolute motion.

Consider the action that transforms the red object in the first demonstration; we can represent this as a RECENTRE(CENTRE : $(1, 1)$) or a MOVE(AMOUNT : $(5.5, -1.5)$). Both are correct and equally valid, but abduction demands we choose the simpler explanation of this mapping. Here, we propose simplicity is determined by the relative frequency of the actions; in other words, recurring operations and parameters have greater explanatory power. The MOVE(AMOUNT : $(5.5, -1.5)$) action explains only one of eight total output objects, but the RECENTRE(CENTRE : $(1, 1)$) action explains two—the upper right corner object in both demonstrations. In this sense, an action set containing RECENTRE(CENTRE : $(-1, 1)$), RECENTRE(CENTRE : $(1, 1)$), RECENTRE(CENTRE : $(-1, -1)$), and RECENTRE(CENTRE : $(1, -1)$) offers a simpler explanation of the observations than any action set based on MOVE could. The RECENTRE operation with possible parameters $(-1, 1)$, $(1, 1)$, $(-1, -1)$, and $(1, -1)$ accounts for all observations most cohesively.

Constructing the simplest action set explaining all demonstrations can be cast as solving the minimum hitting set problem: given a set $\mathcal{S}$ of K partial action sets $\mathcal{A}_k$ taken from all possible actions $\mathcal{U}$, $\mathcal{S} = \{\mathcal{A}_k \mid \mathcal{A}_k \subseteq \mathcal{U}, \ k = 1 \ldots K\}$, find the action set $\mathcal{A}$ hitting each partial action set in $\mathcal{S}$, $\mathcal{A} \subseteq \mathcal{U}$ and $\mathcal{A} \cap \mathcal{A}_k \neq \emptyset \ \ \forall k = 1 \ldots K$, with minimal cost according to some function f. Here, the cost function penalizes each additional operation and parameter introduced; in other words, simple action sets use the least number of operations and distinct operation-parameter compositions. The hitting set problem is NP-Complete, but can be exactly solved quickly in practice.

The demonstration abduction process involves iteratively attempting object hypotheses and finding the simplest action set to explain all demonstrations. This requires intelligently traversing the object hypothesis space, inferring which input object explains each output object, and inferring what actions can perform each input-output object transform. In humans, these search processes are guided by System 1 heuristics. Here, leveraging VSAs, we model such heuristics as rapid computations on the object representations. These heuristics are fast and simple, with strong but imperfect accuracy. We propose VSA-based heuristics for each of these three tasks.

The first heuristic proposes an intelligent order to traverse the object hypothesis space. Instead of randomly trying object hypotheses, our solver first tries the most promising hypotheses. Fundamentally, a useful object hypothesis produces a simple grid representation while enabling simple input-output object transforms. Simple grid representations have as few objects as possible, and simple input-output object transforms are easiest when each output object is uniquely similar to one input object. This heuristic initially encodes objects according to all six hypotheses (see Table 3) and, for each output object, examines the distribution of the similarities to its possible corresponding input objects. We take the softmax of each similarity vector, padded with 0 and 1 values to ensure there are multiple elements, and consider the resultant vector's maximum value. The higher this value, the fewer competing input objects; each additional input object will have some finite similarity to the output object, reducing the value of the maximum softmax

Table 3: Our solver's object hypotheses.

Hypothesis	Description
8-CONNECTED	Each group of 8-connected pixels (i.e., pixels sharing an edge or corner) of the same colour is an object.
4-CONNECTED	Each group of 4-connected pixels (i.e., pixels sharing an edge) of the same colour is an object.
VERTICAL	Each group of vertically contiguous pixels of the same colour is an object.
HORIZONTAL	Each group of horizontally contiguous pixels of the same colour is an object.
COLOUR	All pixels of the same colour, regardless of spatial contiguity, are an object.
PIXEL	Each non-zero (coloured) pixel is its own object.

score and penalizing object hypotheses that create too many objects. Additionally, the higher this value, the more similar one input object is compared to all others; input objects with close similarities have close softmax values, preventing one singular high softmax score and penalizing object hypotheses that produce ambiguous object mappings. Thus, higher values indicate better object hypotheses that enable clear transforms of few objects. Object hypotheses are ranked by their average maximum softmax value and considered by our solver in descending order.

The second heuristic narrows the search space of object-object correspondences. Instead of considering how each input object can transform into each output object, our solver only considers actions applied to the input object most similar to the output object. Dissimilar objects usually cannot be connected by simple transforms, so this heuristic eliminates nonsensical action hypotheses.

The third heuristic narrows the search space of actions. Instead of considering how each operation can transform the input object into the output object, our solver considers only those operations affecting the least similar property of the objects. Actions need only alter dissimilar object properties, so this heuristic removes useless operations from search.

After this stage (see Algorithm 1), our solver has specific hypotheses about what constitutes a task-relevant object and what actions permit simple transformations in all demonstrations.

3.2.2 Rule Induction

Second, our solver induces general object-centric rules explaining the abduced actions. Finding isolated actions explaining each demonstration individually is important, but misses the task's underlying phenomena. Solvers must develop some transferable understanding of the task because ARC-AGI requires generating the correct output grid for arbitrary valid input grids, demonstration or not. In our solver's object-centric view, this means inferring, by both

---

Algorithm 1 Demonstration Abduction

Require: Demonstrations $\mathcal{D}$
1:   $hypotheses \leftarrow$ OBJECTHYPOTHESISHEURISTIC($\mathcal{D}$)   ▷ Object hypotheses ranked by likelihood
2: **for each** hypothesis $\mathcal{H}$ **in** $hypotheses$ **do**
3:      $\mathcal{S} \leftarrow \emptyset$ ▷ Set of action sets explaining each output object
4:      **for** $(\mathbf{G}_{I_d}, \mathbf{G}_{O_d}) \in \mathcal{D}$ **do**
5:          $\mathcal{O}_{I_d} \leftarrow$ PERCEIVEOBJECTS($\mathcal{H}, \mathbf{G}_{I_d}$)   ▷ Input objects
6:          $\mathcal{O}_{O_d} \leftarrow$ PERCEIVEOBJECTS($\mathcal{H}, \mathbf{G}_{O_d}$)   ▷ Output objects
7:          **for** $O_j \in \mathcal{O}_{O_d}$ **do**
8:             $O_i \leftarrow$ OBJECTCORRESPONDENCEHEURISTIC($\mathcal{O}_{I_d}, O_j$)   ▷ Corresponding input object
9:             $operations \leftarrow$ OBJECTACTIONHEURISTIC(O_i, O_j)   ▷ Plausible operations for mapping
10:            $\mathcal{A}_k \leftarrow$ GETPARTIALACTIONSET($O_i, O_j, operations$)   ▷ Action set for output object
11:            $\mathcal{S} \leftarrow \mathcal{S} \cup \{\mathcal{A}_k\}$
12:      $\mathcal{A}, cost \leftarrow$ MINIMUMHITTINGSET($\mathcal{S}$)   ▷ Minimal action set and associated cost
13:      **if** $\mathcal{A}$ successful **and** $cost$ acceptable **then**
14:          **return** $\mathcal{H}, \mathcal{A}$

---

search and learning, what features of the input objects cause them to be transformed in each particular way. In other words, our solver must determine when and, possibly, how each operation should be applied.

Our solver's solution program requires a rule for the conditional application of each operation in the action set. As such, our solver must develop a predictor modelling the condition for whether each operation applies and a predictor for each parameter of each operation. Both types of predictors are implemented as NNs because the high-dimensional distributed object representations produced by the VSA are amenable to neural learning. Although we do not use deep learning, these VSA representations act as deep-learned embeddings; the VSA's inductive biases enable these NNs to better discover the abstractions underlying the conditions and learn the mappings determining the parameters.

Neural learning methods, especially over-parametrized NNs in the sample-few regime, are susceptible to overfitting and shortcut learning. Such NNs tend to get distracted by spurious correlations in the data and often fail to learn the simplest decision criteria. Thus, selecting the input features to our solver's predictor NNs is important. In ARC-AGI tasks, both when and how an operation is applied usually depends on only one or two properties of the input object; the NNs modelling these rules should therefore be exposed only to the subset of relevant properties. Consider ARC-AGI task `a61f2674` (see Fig. 4); when and how the RECOLOUR operation applies depends only on an object's shape property and not its colour or centre properties. Here, supplying the predictor NNs with objects' colour and centre properties serves only to impede learning. Although each predictor NN models System 1, discovering which properties are important for each predictor NN is a search task performed by System 2. Fundamentally, useful properties generalize across demonstrations and useless ones do not.

We believe humans determine which properties are useful by carefully verifying their hypotheses on the demonstrations. Our solver models this process using cross-validation. For each demonstration, our solver, using a particular property hypothesis, trains a predictor NN on data from all other demonstrations and tests the predictor NN on data from the held out demonstration. The aggregate performance across all demonstrations measures the generalizability of the property hypothesis; the highest-scoring hypothesis is selected to learn the final condition. This approach works for many tasks, but not all; some tasks require all demonstrations together to fully understand the rules.

Our solver develops an *operation predictor*, $\mathcal{R}_O$, for each rule modelling the condition for whether the corresponding operation applies to an object. Each operation predictor is an NN mapping the representations of certain properties of an input object onto a probability value signifying whether this operation should be applied to this object. Training input data are the representations of the relevant properties of each input object, and the training labels are binary values indicating whether each input object was subject to this operation. When all labels are positive, our solver assumes the operation applies to all input objects and makes the condition vacuous; an NN is only used when the labels are not uniform.

The operation predictor is a single-layer NN with a learned nonlinearity trained by stochastic gradient descent (SGD). This means the NN learns a single weight vector to transform the representation of the relevant properties of an input object with via a dot product. Thus, the weight vector, if normalized, is itself a valid VSA vector. As such, we restrict the weight vector to unit length during training and learn parameters controlling the steepness and threshold of the decision nonlinearity. The learned nonlinearity enables the NN to match exact patterns for conditions based on discrete features and to model fuzzy abstractions for conditions based on high-level features. With this construction, the learned weight vector represents a prototype of objects subject to the operation. The NN works by computing the similarity between its learned prototype and the input object's representation and applying a sigmoid nonlinearity mapping this similarity onto a probability, thereby predicting an input object as more likely to be subject to the relevant operation the more similar it is to the prototype (see Fig. 7). To accelerate training, we initialize the NN weight vector with the superposition of the input object representations weighted by outcome. We have experimented with more complex, deeper learning architectures, but this simple approach is the most interpretable and theoretically-founded.

Our solver also develops *parameter predictors*, $\mathcal{R}_P$, for each parameter of each operation. Each parameter predictor is an NN mapping the representations of certain properties of an input object onto a VSA vector representing the value of the operation's input parameter. Training input data are the representations of the relevant properties of each input object, and the training labels are the representations of the abduced parameters used when each input object was subject to this operation. When all labels are the same, our solver assumes this parameter is the same for all input objects and always predicts this parameter. Additionally, when all labels match one of the input objects' properties, our solver assumes this parameter always matches that property of the input object and simply predicts the parameter to be this property. An NN is only used otherwise.

Each parameter predictor is a single-layer NN trained by SGD. This means the NN learns a single square weight matrix performing a linear mapping from one VSA vector onto another VSA vector. A single linear transform, although simple, is usually sufficient; because of the VSA's nonlinear encoding scheme, linear functions in the embedding space can express complex nonlinear functions in the feature space. With this construction, the workings of this NN
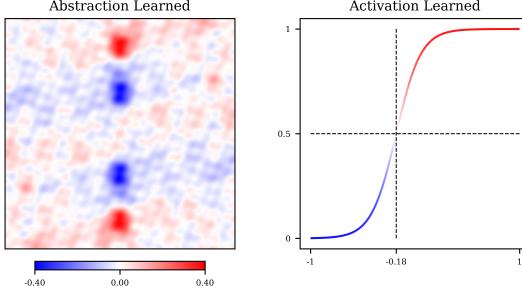
Figure 7: Visualization of the interpretation of the learned RECOLOUR operation predictor NN for ARC-AGI task `a61f2674` (see Fig. 4). The abstraction subfigure (left) shows the prototype learned by the NN, determined here by the input object's shape property. The activation subfigure (right) shows the decision nonlinearity learned by the NN. From these, we can verify the NN selects for short and tall objects. Short objects' shape representations exist only in the centre, producing near-zero similarity mapping onto a likely probability. Medium-height objects' shape representations reach the negative similarity regions indicated by blue, producing a negative similarity mapping onto an unlikely probability. Finally, tall objects' shape representations reach the positive similarity regions indicated by red, producing a net positive similarity mapping onto a likely probability. These representations use $N = 1024$-dimensional vectors.
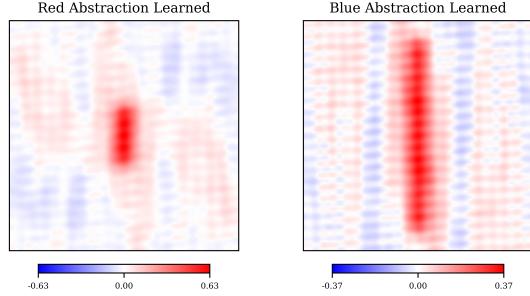

Figure 8: Visualization of the interpretation of the learned RECOLOUR parameter predictor NN for ARC-AGI task `a61f2674` (see Fig. 4). The red subfigure (left) shows the prototype for objects recoloured red learned by the NN, determined here by the input object's shape property. The blue subfigure (right) shows the prototype for objects recoloured blue learned by the NN, again determined here by the input object's shape property. From these, we can verify the NN predicts red for short objects and blue for tall objects. These representations use $N = 1024$-dimensional vectors.

are less interpretable, but we can sometimes cast its operation as a comparison to prototypes too. Given the learned weight matrix, we can find the optimal input representation to most strongly produce a particular output parameter. In this view, the NN works by memorizing the training mappings and, powered by the VSA representations, constructing a reasonable interpolation of the underlying function (see Fig. 8). To accelerate training and bias the NN towards VSA-based solutions, we initialize the NN weights with the average circulant matrix performing the binding operation mapping each input vector to its corresponding label. Again, we have experimented with more complex learning architectures, but this simple approach is the most interpretable and theoretically-founded.

The rule induction process involves, for each operation in the action set, iteratively trying to generalize observations into simple rules based on different object properties. This requires intelligently traversing the object property space. As before, we propose a VSA-based heuristic for this task modelling System 1.

This heuristic proposes an intelligent order to traverse the object property space. Instead of trying to learn from random combinations of object properties, our solver first tries to learn from the most relevant object properties. Fundamentally, a relevant object property, meaning one that is likely to explain the condition or parameter mapping, is usually similar among all objects transformed similarly and different between each pair of objects transformed differently. This heuristic computes the average similarity between the properties of all pairs of objects transformed in the same way and the average similarity between the properties of all pairs of objects transformed in different ways, and takes the difference of their squares. We take the softmax of the resulting vector to capture the relative likelihood of each property explaining the application of each operation and the mapping underlying each parameter. Similar objects are likely to be transformed similarly, so first considering those object properties that are most similar avoids unlikely explanations.

After this stage (see Algorithm 2), our solver has developed a program as a set of rules describing when and how to apply each operation to input objects.

3.2.3   Answer Deduction

Third, our solver deduces the output grid for each query. Any solver must apply whatever understanding it has obtained about the demonstrations to the queries. This requires generating an output grid from scratch.

Algorithm 2 Rule Induction

Require: Input objects $\mathcal{O}_I$, Action set $\mathcal{A}$

1: $\mathcal{P} \leftarrow \emptyset$ ▷ Program comprising a set of rules
2: **for** $operation \in \mathcal{A}$ **do**
3:     $labels \leftarrow$ GETOBJECTOPERATIONLABELS$(\mathcal{A}, operation)$ ▷ Known from abduction stage
4:     $properties \leftarrow$ OBJECTPROPERTYHEURISTIC$(\mathcal{O}_I, labels)$ ▷ Properties ranked by relevance
5:     $\mathcal{R}_O \leftarrow$ LEARNNN$(\mathcal{O}_I, labels, properties)$ ▷ Bypassed if all labels are the same
6:     **for each** $parameter$ required by $operation$ **do**
7:         $labels \leftarrow$ GETOBJECTPARAMETERLABELS$(\mathcal{A}, operation, parameter)$
8:         $properties \leftarrow$ OBJECTPROPERTYHEURISTIC$(\mathcal{O}_I, labels)$
9:         $\mathcal{R}_P \leftarrow$ LEARNNN$(\mathcal{O}_I, labels, properties)$ ▷ Bypassed if all labels are the same or unchanged
10:    $\mathcal{P} \leftarrow \mathcal{P} \cup \{(operation, \mathcal{R}_O, \mathcal{R}_P)\}$
11: **return** $\mathcal{P}$

By our solution structure, this means sizing the output grid and predicting and executing the actions for each input object. Each input object is considered separately by each rule; if the object satisfies the rule's learned condition, then the rule's action, with parameters predicted as needed, is executed and the new object is added to the output grid. Objects may be subject to multiple rules, producing multiple output objects, and rules may be applied to multiple objects.

After this stage (see Algorithm 3), our solver has solved the task.

4 Results

We evaluate the performance of our solver on ARC-AGI and related benchmarks. Because our solver generates task solutions as neurosymbolic programs, we can probe performance by executing the solution program on both the training input grids and the testing input grids. When our solver can correctly solve the demonstrations, it has some understanding of the task; when our solver can correctly solve the queries, it has a true, generalizable understanding of the task. Therefore, demonstration performance indicates whether our solver can conceptualize the objects and actions underlying task transformations, but query performance reveals whether our solver can extract and apply the underlying patterns. We report both *accuracy*, the proportion of all demonstrations or queries in the dataset solved correctly, and *task accuracy*, the proportion of tasks for which all associated demonstrations or queries were solved correctly. The main performance metric, except where noted, is query task accuracy. All experiments use representations with $N = 4096$-dimensional vectors.

4.1 Main Benchmarks

Our solver scores 10.8% on ARC-AGI-1-Train and 3.0% on ARC-AGI-1-Eval (see Table 4). We report performance on the training splits in addition to the evaluation splits because they are more approachable (note that the training splits are not used for training; our solver does not require any pre-training). As expected, our solver performs better on ARC-AGI-1 than on ARC-AGI-2 and performs better on training splits than evaluation splits.

Algorithm 3 Answer Deduction

Require: Object hypothesis $\mathcal{H}$, Program $\mathcal{P}$, Queries $\mathcal{Q}$

1: $results \leftarrow [\,]$ ▷ Test output grids
2: **for** $\mathbf{G}_{I_q} \in \mathcal{Q}$ **do**
3:     $\mathcal{O}_{I_q} \leftarrow$ PERCEIVEOBJECTS$(\mathcal{H}, \mathbf{G}_{I_q})$ ▷ Input objects
4:     $\mathcal{O}_{O_q} \leftarrow \emptyset$ ▷ Output objects
5:     **for** $O_i \in \mathcal{O}_{I_q}$ **do**
6:         **for** $(operation, \mathcal{R}_O, \mathcal{R}_P) \in \mathcal{P}$ **do**
7:             **if** $\mathcal{R}_O(O_i) \geq 50\%$ **then**   ▷ Threshold is arbitrarily 50%
8:                 $\mathcal{O}_{O_q} \leftarrow \mathcal{O}_{O_q} \cup$ EXECUTEACTION$(O_i, operation, \mathcal{R}_P(O_i))$
9:     $results \leftarrow results \cup \mathcal{O}_{O_q}$
10: **return** $results$

Table 4: Our solver's performance on ARC-AGI. We report performance on each benchmark split separately.

Benchmark Split	Demonstrations		Queries	
	Acc. (%)	Task Acc. (%)	Acc. (%)	Task Acc. (%)
ARC-AGI-1-Train ($n = 400$)	57.5	48.8	12.7	10.8
ARC-AGI-1-Eval ($n = 400$)	43.6	33.5	3.3	3.0
ARC-AGI-2-Train ($n = 1000$)	46.7	35.9	6.5	5.8
ARC-AGI-2-Eval ($n = 120$)	19.5	11.7	0.0	0.0

4.2 Related Benchmarks

Because of ARC-AGI's extreme difficulty, many simpler versions capturing certain important features of the benchmark have been made. We also evaluate the performance of our solver on some of these benchmarks.

4.2.1 Sort-of-ARC

Sort-of-ARC (Assouel et al., 2022) is a collection of 1000 restricted ARC-AGI-like tasks. Each task comprises $|\mathcal{D}| = 5$ demonstrations and $|\mathcal{Q}| = 1$ query; all grids are 20×20 and contain three non-overlapping objects of random colour, location, and 3×3 shape. The solution is always to move all objects matching a randomly-chosen colour or shape by one pixel in one of the four cardinal directions and leave all other objects unchanged. Although simple, Sort-of-ARC directly tests condition-action learning.

Our solver scores 94.5% on Sort-of-ARC (see Table 5). Notably, our solver performs better on tasks with a colour-based condition than those with a shape-based condition. This is because different colour representations have near-zero similarity but different shape representations may have very high similarity. As a result, without diverse negative examples in the demonstrations, the operation predictors tend to over-generalize shape abstractions (e.g., our solver may move U-shaped objects when the solution is to move all O-shaped objects and the demonstrations lack U-shaped or similarly-shaped objects). This generalization capacity enables our solver to solve more complex tasks, but sometimes poses challenges for simple tasks requiring strict rules.

4.2.2 1D-ARC

1D-ARC (Xu et al., 2024) is a collection of 900 one-dimensional ARC-AGI-like tasks. Each task comprises $|\mathcal{D}| = 3$ demonstrations and $|\mathcal{Q}| = 1$ query; all grids are a single row of pixels. Tasks are divided into 18 types, such as "Move Dynamic" and "Recolour By Size", each capturing some important operation in ARC-AGI. Each type contains 50 random instantiations of the operation. Designed to probe how LLMs solve ARC-AGI, 1D-ARC thoroughly tests the application of particular operations.

Our solver scores 83.1% on 1D-ARC (see Table 6). Our solver can consistently solve tasks involving operations within its DSL, such as those requiring recolouring (e.g., Recolour by Odd Even and Recolour by Size), recentring (e.g., Move 1, Move Dynamic, and Scaling), and reshaping (e.g., Fill, Hollow, and Pattern Copy). But, our solver falters on tasks requiring operations outside of its DSL, such as reflecting (e.g., Flip and Mirror). Still, our solver vastly outperforms GPT-4 (OpenAI et al., 2024) on this benchmark at a tiny fraction of the computational cost.

Table 5: Our solver's performance on Sort-of-ARC (Assouel et al., 2022). We report performance on tasks with a colour-based condition and tasks with a shape-based condition separately. Our version of the dataset was randomly generated according to Assouel et al.'s (2022) description.

Benchmark Split	Demonstrations		Queries	
	Acc. (%)	Task Acc. (%)	Acc. (%)	Task Acc. (%)
All ($n = 1000$)	99.7	99.4	—	94.5
Colour ($n = 500$)	100.0	100.0	—	100.0
Shape ($n = 500$)	99.4	98.8	—	89.0

Table 6: Our solver's performance on 1D-ARC (Xu et al., 2024). We report performance on each task type separately. We compare our results to results obtained by directly prompting GPT-4 (OpenAI et al., 2024), as reported in the original work (Xu et al., 2024).

Benchmark Split	Demonstrations		Queries		GPT-4
	Acc. (%)	Task Acc. (%)	Acc. (%)	Task Acc. (%)	
All ($n = 900$)	97.1	92.9	—	83.1	—
Move 1 ($n = 50$)	100.0	100.0	—	**100.0**	66.0
Move 2 ($n = 50$)	100.0	100.0	—	**100.0**	26.0
Move 3 ($n = 50$)	100.0	100.0	—	**100.0**	24.0
Move Dynamic ($n = 50$)	100.0	100.0	—	**90.0**	22.0
Move 2 Towards ($n = 50$)	100.0	100.0	—	**98.0**	34.0
Fill ($n = 50$)	100.0	100.0	—	**100.0**	66.0
Padded Fill ($n = 50$)	100.0	100.0	—	**46.0**	26.0
Hollow ($n = 50$)	100.0	100.0	—	**100.0**	56.0
Flip ($n = 50$)	77.3	44.0	—	12.0	**70.0**
Mirror ($n = 50$)	94.0	90.0	—	**28.0**	20.0
Denoise ($n = 50$)	100.0	100.0	—	**100.0**	36.0
Denoise Multicolour ($n = 50$)	100.0	100.0	—	**98.0**	60.0
Pattern Copy ($n = 50$)	100.0	100.0	—	**100.0**	36.0
Pattern Copy Multicolour ($n = 50$)	100.0	100.0	—	**100.0**	38.0
Recolour by Odd Even ($n = 50$)	100.0	100.0	—	**96.0**	32.0
Recolour by Size ($n = 50$)	100.0	100.0	—	**100.0**	28.0
Recolour by Size Comparison ($n = 50$)	76.0	38.0	—	**32.0**	20.0
Scaling ($n = 50$)	100.0	100.0	—	**96.0**	88.0

4.2.3 KidsARC

KidsARC (Opiełka et al., 2024) is a collection of 17 single-demonstration ARC-AGI-like tasks. Each task comprises $|\mathcal{D}| = 1$ demonstration and $|\mathcal{Q}| = 1$ query; all grids in the 9 simple tasks are 3×3, and all grids in the 8 concept tasks are 5×5. The 9 simple tasks invoke elementary concepts and can be presented as $A : B :: C : D$ or as $A : C :: B : D$; the 8 concept tasks invoke higher-level abstractions. Designed to understand how children approach ARC-AGI, KidsARC uniquely tests extreme generalization.

Our solver scores 57.7% on KidsARC (see Table 7). Our solver can abduce the correct operations (e.g., recolouring, recentring, and reshaping) and even learn useful abstractions from a single demonstration (e.g., objects near the top of the grid and dominant versus noise objects). But, our solver falters when extreme generalization of a high-level concept is needed from a single example (e.g., objects between other objects and objects with the majority colour). Again, our solver outperforms many LLMs on this benchmark, still at a tiny fraction of the computational cost.

4.2.4 ConceptARC

ConceptARC (Moskvichev et al., 2023) is a collection of 176 complex ARC-AGI-like tasks. Each task comprises a variable number of demonstrations and $|\mathcal{Q}| = 3$ queries; like ARC-AGI, grid sizes are variable. Tasks are divided into

Table 7: Our solver's performance on KidsARC (Opiełka et al., 2024). We report performance on simple and concept tasks separately. We compare our results to aggregate results obtained by directly prompting various LLMs, including GPT-4 (OpenAI et al., 2024), as reported in the original work (Opiełka et al., 2024).

Benchmark Split	Demonstrations		Queries		LLMs
	Acc. (%)	Task Acc. (%)	Acc. (%)	Task Acc. (%)	
All ($n = 26$)	—	92.3	—	57.7	—
Simple ($n = 18$)	—	94.4	—	**66.7**	33.2
Concept ($n = 8$)	—	87.5	—	**37.5**	11.9

Table 8: Our solver's performance on ConceptARC (Moskvichev et al., 2023). We report performance on the minimal tasks as well as each task type separately. We compare our results to results obtained by directly prompting GPT-4 (OpenAI et al., 2024), as reported in the original work (Moskvichev et al., 2023). Note that ConceptARC measures performance as the proportion of queries solved correctly, not the proportion of tasks with all queries solved correctly.

Benchmark Split	Demonstrations		Queries		GPT-4
	Acc. (%)	Task Acc. (%)	Acc. (%)	Task Acc. (%)	
All ($n = 176$)	79.1	73.3	20.5	11.4	—
Minimal ($n = 16$)	91.3	87.5	52.1	31.2	—
Above and Below ($n = 10$)	87.5	90.0	**26.7**	20.0	23.3
Center ($n = 10$)	80.0	70.0	23.3	10.0	**33.3**
Clean Up ($n = 10$)	91.3	90.0	6.7	0.0	**20.0**
Complete Shape ($n = 10$)	90.5	90.0	3.3	0.0	**23.3**
Copy ($n = 10$)	65.2	70.0	0.0	0.0	**23.3**
Count ($n = 10$)	11.1	10.0	3.3	0.0	**13.3**
Extend To Boundary ($n = 10$)	93.1	90.0	3.3	0.0	**6.7**
Extract Objects ($n = 10$)	43.5	40.0	**20.0**	20.0	3.3
Filled and Not Filled ($n = 10$)	82.8	70.0	**20.0**	0.0	16.7
Horizontal and Vertical ($n = 10$)	96.0	90.0	**36.7**	30.0	26.7
Inside and Outside ($n = 10$)	69.0	60.0	**20.0**	10.0	10.0
Move To Boundary ($n = 10$)	92.0	90.0	**30.0**	10.0	20.0
Order ($n = 10$)	76.2	70.0	10.0	0.0	**26.7**
Same and Different ($n = 10$)	84.8	70.0	6.7	0.0	**16.7**
Top and Bottom 2D ($n = 10$)	94.1	80.0	**60.0**	50.0	23.3
Top and Bottom 3D ($n = 10$)	80.6	70.0	6.7	0.0	**20.0**

16 concept groups, such as "Above and Below" and "Same and Different", each capturing some important abstraction in ARC-AGI. Each group contains a minimal example alongside 10 carefully-designed instantiations of the concept. Designed to probe how completely ARC-AGI solvers understand the concepts they use, ConceptARC thoroughly tests the comprehension of particular concepts.

Our solver scores 20.5% on ConceptARC (see Table 8). Our solver performs well on spatial concepts (e.g., Above and Below, Filled and Not Filled, Horizontal and Vertical, and Top and Bottom 2D) because of the inductive biases provided by SSP representations. But, our solver falters on tasks requiring interpolative object perception (e.g., Clean Up, Complete Shape, Copy, and Top and Bottom 3D) and advanced explicit reasoning (e.g., Count, Order, and Same and Different). Still, our solver performs comparably to GPT-4 (OpenAI et al., 2024) on this benchmark, again at a tiny fraction of the computational cost.

4.2.5 MiniARC

MiniARC (Kim et al., 2022) is a collection of 149 miniature ARC-AGI-like tasks. Each task comprises a variable number of demonstrations and $|\mathcal{Q}| = 1$ query; all grids are 5×5. Like ARC-AGI, task content is unconstrained. Designed to simplify ARC-AGI without compromising its core tenets, MiniARC tests reasoning without grid size effects.

Our solver scores 13.4% on MiniARC (see Table 9). Notably, our solver can solve multiple tasks requiring conceptualizing and implementing arbitrary object mappings (e.g., tasks `l6aftgp22mlspnm1zxb`, `l6acypud56b4m1z5409`, and `l6afcchh0wfew0rhswzq`).

Table 9: Our solver's performance on MiniARC (Kim et al., 2022).

Benchmark Split	Demonstrations		Queries	
	Acc. (%)	Task Acc. (%)	Acc. (%)	Task Acc. (%)
All ($n = 149$)	69.9	55.0	—	13.4

5 Discussion

Our solver structures solutions as a specific definition of task-relevant objects and a specific program comprising a set of rules describing how objects behave. Objects are represented by high-dimensional vectors encoding their colour, centre, and shape. Program rules are if-then statements that conditionally execute standard actions on certain objects based on object features. This paradigm is flexible enough to express natural solutions to many ARC-AGI tasks, but simple enough for synthesis to remain tractable.

Our solver generates solutions with a heuristic-guided search process and a neurosymbolic learning architecture. First, our solver abduces the simplest objects and actions explaining the demonstrations. Second, our solver generalizes its observations into rules applicable to any valid input grid. Third, our solver applies these rules to answer all queries. This process combines the strengths of symbolic and connectionist AI, leveraging VSA representations for robustness and interpretability in search and learning.

5.1 Strengths

Any model of human intelligence must be efficient and interpretable. For ARC-AGI, this is true of both the solution synthesis process and the synthesized solution itself.

Human intelligence is efficient. The brain's power consumption is much lower than that of modern computers, and humans effortlessly solve ARC-AGI. Because our solver uses neither brute force search nor large NNs, its computational demands are relatively small. Our solver considers few solution candidates in depth because simple and cheap heuristics are used to prune the search space, minimizing wasted energy on unnecessary reasoning. Overall, our solver is efficient.

Human intelligence is interpretable. When prompted, humans can detail both the problem solving steps they took and the solution they arrived at. Because intermediate hypotheses are few, all steps taken by our solver are traceable and intentional. The solution programs generated are compact and integrate directly interpretable symbolic expressions with interpretable NNs implementing simple operations on structured data. Overall, our solver is interpretable.

Our methods are applicable to other problems beyond ARC-AGI. We use general VSA representations capable of implementing many features of cognition. For example, Spaun (Choo, 2018; Eliasmith et al., 2012), the world's largest functional brain model, uses similar representations to closely model several different parts of the brain and perform a wide array of cognitive tasks. Additionally, we apply general principles learned from human psychology to motivate and structure our solver. For example, we explicitly model the interplay of System 1 intuitions and System 2 reasoning (Kahneman, 2011) in a unified substrate. Our particular implementation and architecture is ARC-AGI-specific, but the underlying principles and tools are general; similar VSA-based approaches may be taken to solve other problems requiring System 1, System 2, or both. Fundamentally, VSA-powered neurosymbolic models represent a universal approach to cognitive modelling and AI.

Our approach furthers progress on ARC-AGI. Chollet, the creator of ARC-AGI, has argued the proper solution to ARC-AGI may be neurosymbolic (Chollet et al., 2025a), involving both symbolic program synthesis and connectionist deep learning. VSAs offer a new means of bridging symbolic and connectionist AI for ARC-AGI. To the best of our knowledge, our solver is the first to apply VSAs to ARC-AGI, making our approach unique and novel. Additionally, because we use cognitively and biologically plausible representations in a psychologically-motivated framework free from brute force search and opaque deep NNs, we believe our solver is the most cognitively plausible one yet.

Our solver owes its success to the same strengths of VSAs that make them so effective for other cognitive tasks. First, VSAs bridge symbolicism and connectionism, combining the benefits of both. In a single unified framework, VSAs enable both heuristic-guided search with explicit reasoning and sample-efficient neural learning. Second, VSAs capture many inductive biases humans have. In this sense, VSAs effectively model human intelligence, which remains the best ARC-AGI solver. The required numbers and geometry core knowledge priors are implicitly provided by SSP representations, and objectness is explicitly modelled by our solver in other ways (we leave goal-directedness for future work).

5.2 Limitations

Our solver is primarily limited by its inability to generate new grid size change, object, and action hypotheses on-the-fly. The demonstration abduction stage is only capable of applying existing notions of what grid sizes, objects, and actions may be taken from finite sets. This is permissible with a sufficiently-sized hypothesis set because all ARC-AGI tasks must be derived from the core knowledge priors, but our solver's sets are currently not large enough. Furthermore, we do not address the fundamental problem of how these conceptualizations came to be; instead, we

assume they have already been acquired. Maintaining cognitive plausibility, we model the process by which humans recombine existing knowledge, represented as inductive biases built into the VSAs and symbolic programs, to solve new tasks. In this sense, our solver is not artificial *general* intelligence because it uses a domain-*specific* language; a true AGI solution must be DSL-open, discovering and applying novel transformations on-the-fly.

Other limitations are more pragmatic. First, our solver cannot apply chained operations to a single persistent input object. This simplifies solution synthesis by narrowing the search space of object-to-object transforms at the cost of preventing arbitrary compositionality of operations, but the GENERATE operation can compose RECOLOUR, RECENTRE, and SHAPE operations. Second, our solver cannot apply multiple instantiations of one operation to the same input object. This simplifies program representation by restricting the program to one rule for each operation at the cost of solving certain tasks. Third, our solver can conceptualize neither many-to-one nor many-to-many object mappings. This simplifies solution synthesis also by narrowing the search space of object-to-object transforms at the cost of solving tasks requiring comparison between objects, but creative object hypotheses can often circumvent this. Fourth, our solver cannot conceptualize multi-coloured objects. This simplifies object representation by allowing separate colour, centre, and shape properties at the cost of solving certain tasks, but creative actions can often circumvent this. Fifth, our solver cannot conceptualize conditions of high-level object properties. This simplifies object representation and maintains cognitive plausibility at the cost of solving certain tasks, but these properties, such as exact size and symmetry, can often be well-approximated by other object properties, such as shape. All of these limitations are the subject of ongoing work.

6   Conclusion

We presented a novel, neurosymbolic, cognitively plausible ARC-AGI solver. Our solver works by VSA-enabled object-centric program synthesis. Grids are represented as collections of objects encoded with a VSA, and solutions are represented as programs comprising a set of rules implemented with a VSA. Solution synthesis uses VSAs to bridge symbolic and connectionist AI; modelling human intelligence, VSA-powered System-1-style heuristics guide a System-2-style intentional search process incorporating abductive, inductive, and deductive reasoning. VSA representations enable sample-efficient neural learning of rules, integrating automated abstraction discovery into the reasoning process. The solution generation process is both efficient and interpretable, maintaining cognitive plausibility. Our solver shows some initial success, scoring 10.8% on ARC-AGI-1-Train and 3.0% on ARC-AGI-1-Eval. Although these results are not state-of-the-art, they indicate VSAs hold promise for ARC-AGI and warrant further investigation. Importantly, our approach is unique; we believe we are the first to apply VSAs to ARC-AGI and, in doing so, have developed the most cognitively plausible ARC-AGI solver yet.

Acknowledgements

References

Acquaviva, S., Pu, Y., Kryven, M., Sechopoulos, T., Wong, C., Ecanow, G., Nye, M., Tessler, M., and Tenenbaum, J. (2022). Communicating natural programs to humans and machines. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A., editors, *Advances in Neural Information Processing Systems*, volume 35, pages 3731–3743. Curran Associates, Inc.

Akyürek, E., Damani, M., Zweiger, A., Qiu, L., Guo, H., Pari, J., Kim, Y., and Andreas, J. (2025). The surprising effectiveness of test-time training for few-shot learning. In *Forty-second International Conference on Machine Learning*.

Assouel, R., Rodriguez, P., Taslakian, P., Vazquez, D., and Bengio, Y. (2022). Object-centric compositional imagination for visual abstract reasoning. In *ICLR 2022 Workshop on the Elements of Reasoning: Objects, Structure and Causality*.

Banburski, A., Gandhi, A., Alford, S., Dandekar, S., Chin, S., and Poggio, T. (2020). Dreaming with ARC. In *Learning Meets Combinatorial Algorithms at NeurIPS2020*.

Bednarek, J. and Krawiec, K. (2024). Learning to solve abstract reasoning problems with neurosymbolic program synthesis and task generation. In Besold, T. R., d'Avila Garcez, A., Jimenez-Ruiz, E., Confalonieri, R., Madhyastha, P., and Wagner, B., editors, *Neural-Symbolic Learning and Reasoning*, pages 386–402. Springer Nature Switzerland.

Bengio, Y., LeCun, Y., and Hinton, G. E. (2021). Deep learning for AI. *Communications of the ACM*, 64(7):58–65.

Berglund, L., Tong, M., Kaufmann, M., Balesni, M., Stickland, A. C., Korbak, T., and Evans, O. (2024). The reversal curse: LLMs trained on "A is B" fail to learn "B is A". In *The Twelfth International Conference on Learning Representations*.

Bober-Irizar, M. and Banerjee, S. (2024). Neural networks for abstraction and reasoning. *Scientific Reports*, 14(27823).

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Chollet, F. (2019). On the measure of intelligence. https://arxiv.org/abs/1911.01547.

Chollet, F., Knoop, M., Kamradt, G., and Landers, B. (2025a). ARC Prize 2024: Technical report. https://arxiv.org/abs/2412.04604.

Chollet, F., Knoop, M., Kamradt, G., Landers, B., and Pinkard, H. (2025b). ARC-AGI-2: A new challenge for frontier AI reasoning systems. https://arxiv.org/abs/2505.11831.

Chollet, F., Knoop, M., Kamradt, G., Reade, W., and Howard, A. (2025c). ARC Prize 2025. https://kaggle.com/competitions/arc-prize-2025. Kaggle.

Chollet, F., Knoop, M., Landers, B., Kamradt, G., Jud, H., Reade, W., and Howard, A. (2024). ARC Prize 2024. https://kaggle.com/competitions/arc-prize-2024. Kaggle.

Choo, X. (2010). The ordinal serial encoding model: Serial memory in spiking neurons. Master's thesis, University of Waterloo.

Choo, X. (2018). *Spaun 2.0: Extending the World's Largest Functional Brain Model*. PhD thesis, University of Waterloo.

Cole, J. and Osman, M. (2025). Don't throw the baby out with the bathwater: How and why deep learning for ARC. https://arxiv.org/abs/2506.14276.

Crick, F. (1989). The recent excitement about neural networks. *Nature*, 337:129–132.

d'Avila Garcez, A. and Lamb, L. C. (2023). Neurosymbolic AI: The 3rd wave. *Artificial Intelligence Review*, 56(11):12387–12406.

de Carvalho, G. H., Knap, O., and Pollice, R. (2025). Show, don't tell: Evaluating large language models beyond textual understanding with ChildPlay. https://arxiv.org/abs/2407.11068.

Doeller, C. F., Barry, C., and Burgess, N. (2010). Evidence for grid cells in a human memory network. *Nature*, 463:657–661.

Dumont, N. S.-Y. (2025). *Symbols, Dynamics, and Maps: A Neurosymbolic Approach to Spatial Cognition*. PhD thesis, University of Waterloo.

Dumont, N. S.-Y. and Eliasmith, C. (2020). Accurate representation for spatial cognition using grid cells. In *42nd Annual Meeting of the Cognitive Science Society*, pages 2367–2373, Toronto, ON. Cognitive Science Society.

Dumont, N. S.-Y., Furlong, P. M., Orchard, J., and Eliasmith, C. (2023). Exploiting semantic information in a spiking neural SLAM system. *Frontiers in Neuroscience*, 17.

Dumont, N. S.-Y., Orchard, J., and Eliasmith, C. (2022). A model of path integration that connects neural and symbolic representation. In *44th Annual Meeting of the Cognitive Science Society*, pages 3662–3668, Toronto, ON. Cognitive Science Society.

Eliasmith, C. (2013). *How to Build a Brain: A Neural Architecture for Biological Cognition*. Oxford Series on Cognitive Models and Architectures. Oxford University Press.

Eliasmith, C. and Anderson, C. H. (2003). *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems*. MIT Press.

Eliasmith, C., Stewart, T. C., Choo, X., Bekolay, T., DeWolf, T., Tang, Y., and Rasmussen, D. (2012). A large-scale model of the functioning brain. *Science*, 338(6111):1202–1205.

Fedorenko, E., Piantadosi, S. T., and Gibson, E. A. F. (2024). Language is primarily a tool for communication rather than thought. *Nature*, 630:575–586.

Ferré, S. (2024). Tackling the Abstraction and Reasoning Corpus (ARC) with object-centric models and the MDL principle. In Miliou, I., Piatkowski, N., and Papapetrou, P., editors, *Advances in Intelligent Data Analysis XXII*, pages 3–15, Cham. Springer Nature Switzerland.

Fletcher-Hill, P. (2024). Mini-ARC: Solving abstraction and reasoning puzzles with small transformer models. `https://www.paulfletcherhill.com/mini-arc.pdf`.

Fodor, J. A. and Pylyshyn, Z. W. (1988). Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1):3–71.

Frady, E. P., Kleyko, D., Kymn, C. J., Olshausen, B. A., and Sommer, F. T. (2022). Computing on functions using randomized vector representations (in brief). In *Proceedings of the 2022 Annual Neuro-Inspired Computational Elements Conference*, NICE '22, pages 115–122, New York, NY, USA. Association for Computing Machinery.

Franzen, D., Disselhoff, J., and Hartmann, D. (2025). Product of experts with LLMs: Boosting performance on ARC is a matter of perspective. In Singh, A., Fazel, M., Hsu, D., Lacoste-Julien, S., Berkenkamp, F., Maharaj, T., Wagstaff, K., and Zhu, J., editors, *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 17657–17671. PMLR.

Fu, T., Ferrando, R., Conde, J., Arriaga, C., and Reviriego, P. (2024). Why do large language models (LLMs) struggle to count letters? `https://arxiv.org/abs/2412.18626`.

Furlong, P. M. and Eliasmith, C. (2022). Fractional binding in vector symbolic architectures as quasi-probability statements. In *44th Annual Meeting of the Cognitive Science Society*. Cognitive Science Society.

Furlong, P. M. and Eliasmith, C. (2023). Bridging cognitive architectures and generative models with vector symbolic algebras. In *Proceedings of the 2023 AAAI Fall Symposium*.

Galanti, L. and Baron, E. (2024). Intelligence analysis of language models. `https://arxiv.org/abs/2407.18968`.

Garnelo, M. and Shanahan, M. (2019). Reconciling deep learning with symbolic artificial intelligence: representing objects and relations. *Current Opinion in Behavioral Sciences*, 29:17–23. Artificial Intelligence.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2015). Explaining and harnessing adversarial examples. `https://arxiv.org/abs/1412.6572`.

Greff, K., van Steenkiste, S., and Schmidhuber, J. (2020). On the binding problem in artificial neural networks. `https://arxiv.org/abs/2012.05208`.

Hafting, T., Fyhn, M., Molden, S., Moser, M.-B., and Moser, E. I. (2005). Microstructure of a spatial map in the entorhinal cortex. *Nature*, 436:801–806.

Harnad, S. (1990). The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42(1):335–346.

Hocquette, C. and Cropper, A. (2025). Relational decomposition for program synthesis. In Kwok, J., editor, *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI-25*, pages 4526–4534. International Joint Conferences on Artificial Intelligence Organization.

Hodel, M. (2023). Domain specific language for the Abstraction and Reasoning Corpus (ARC-DSL). `https://github.com/michaelhodel/arc-dsl/`.

Hodel, M. (2024). Addressing the Abstraction and Reasoning Corpus via procedural example generation. `https://arxiv.org/abs/2404.07353`.

Johnson, A., Vong, W. K., Lake, B. M., and Gureckis, T. M. (2021). Fast and flexible: Human program induction in abstract reasoning tasks. In *43rd Annual Meeting of the Cognitive Science Society*. Cognitive Science Society.

Kahneman, D. (2011). *Thinking, Fast and Slow*. Farrar, Straus, and Giroux.

Kautz, H. (2022). The third AI summer: AAAI Robert S. Engelmore memorial lecture. *AI Magazine*, 43(1):105–125.

Kim, S., Phunyaphibarn, P., Ahn, D., and Kim, S. (2022). Playgrounds for abstraction and reasoning. In *NeurIPS 2022 Workshop on Neuro Causal and Symbolic AI (nCSI)*.

Komer, B. (2020). *Biologically Inspired Spatial Representation*. PhD thesis, University of Waterloo.

Komer, B., Stewart, T. C., Voelker, A. R., and Eliasmith, C. (2019). A neural representation of continuous space using fractional binding. In *41st Annual Meeting of the Cognitive Science Society*, Montreal, QC. Cognitive Science Society.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Proceedings of the 26th International Conference on Neural Information Processing Systems – Volume 1*, NIPS'12, pages 1097–1105, Red Hook, NY, USA. Curran Associates Inc.

Lampinen, A. K., Dasgupta, I., Chan, S. C. Y., Sheahan, H. R., Creswell, A., Kumaran, D., McClelland, J. L., and Hill, F. (2024). Language models, like humans, show content effects on reasoning tasks. *PNAS Nexus*, 3(7).

Laube, R. (2024). QAVSA: Question answering using vector symbolic algebras. Master's thesis, University of Waterloo.

LeCun, Y., Bengio, Y., and Hinton, G. E. (2015). Deep learning. *Nature*, 521:436–444.

Lee, S., Sim, W., Shin, D., Seo, W., Park, J., Lee, S., Hwang, S., Kim, S., and Kim, S. (2025). Reasoning abilities of large language models: In-depth analysis on the Abstraction and Reasoning Corpus. *ACM Transactions on Intelligent Systems and Technology*.

LeGris, S., Vong, W. K., Lake, B. M., and Gureckis, T. M. (2024). H-ARC: A robust estimate of human performance on the Abstraction and Reasoning Corpus benchmark. `https://arxiv.org/abs/2409.01374`.

LeGris, S., Vong, W. K., Lake, B. M., and Gureckis, T. M. (2025). A comprehensive behavioral dataset for the Abstraction and Reasoning Corpus. *Scientific Data*, 12(1380).

Li, W.-D., Hu, K., Larsen, C., Wu, Y., Alford, S., Woo, C., Dunn, S. M., Tang, H., Zheng, W.-L., Pu, Y., and Ellis, K. (2025). Combining induction and transduction for abstract reasoning. In *The Thirteenth International Conference on Learning Representations*.

Lim, M., Lee, S., Abitew, L. W., and Kim, S. (2024). Abductive symbolic solver on Abstraction and Reasoning Corpus. `https://arxiv.org/abs/2411.18158`.

Lindsay, R. K., Buchanan, B. G., Feigenbaum, E. A., and Lederberg, J. (1993). DENDRAL: A case study of the first expert system for scientific hypothesis formation. *Artificial Intelligence*, 61(2):209–261.

Macfarlane, M. V. and Bonnet, C. (2025). Searching latent program spaces. `https://arxiv.org/abs/2411.08706`.

Marcus, G. (2020). The next decade in AI: Four steps towards robust artificial intelligence. `https://arxiv.org/abs/2002.06177`.

McCoy, R. T., Yao, S., Friedman, D., Hardy, M. D., and Griffiths, T. L. (2024). Embers of autoregression show how large language models are shaped by the problem they are trained to solve. *Proceedings of the National Academy of Sciences*, 121(41).

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133.

Minsky, M. (1968). *Semantic Information Processing*. The MIT Press.

Mitchell, M., Palmarini, A. B., and Moskvichev, A. K. (2023). Comparing humans, GPT-4, and GPT-4V on abstraction and reasoning tasks. In *AAAI 2024 Workshop on "Are Large Language Models Simply Causal Parrots?"*.

Moskvichev, A. K., Odouard, V. V., and Mitchell, M. (2023). The ConceptARC benchmark: Evaluating understanding and generalization in the ARC domain. *Transactions on Machine Learning Research*.

Nam, A. J. and McClelland, J. L. (2024). Systematic human learning and generalization from a brief tutorial with explanatory feedback. *Open Mind*, 8:148–176.

Nawaz, U., Anees-ur-Rahaman, M., and Saeed, Z. (2025). A review of neuro-symbolic AI integrating reasoning and learning for advanced cognitive systems. *Intelligent Systems with Applications*, 26.

Neumann, N. and Pintér, A. (2023). Solving ARC with non-procedural program induction. In *2023 IEEE 23rd International Symposium on Computational Intelligence and Informatics (CINTI)*, pages 271–276.

Newell, A. (1980). Physical symbol systems. *Cognitive Science*, 4(2):135–183.

Newell, A. and Simon, H. A. (1976). Computer science as empirical inquiry: symbols and search. *Communications of the ACM*, 19(3):113–126.

OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., Avila, R., Babuschkin, I., Balaji, S., Balcom, V., Baltescu, P., Bao, H., Bavarian, M., Belgum, J., Bello, I., Berdine, J., Bernadett-Shapiro, G., Berner, C., Bogdonoff, L., Boiko, O., Boyd, M., Brakman, A.-L., Brockman, G., Brooks, T., Brundage, M., Button, K., Cai, T., Campbell, R., Cann, A., Carey, B., Carlson, C., Carmichael, R., Chan, B., Chang, C., Chantzis, F., Chen, D., Chen, S., Chen, R., Chen, J., Chen, M., Chess, B., Cho, C., Chu, C., Chung, H. W., Cummings, D., Currier, J., Dai, Y., Decareaux, C., Degry, T., Deutsch, N., Deville, D., Dhar, A., Dohan, D., Dowling, S., Dunning, S., Ecoffet, A., Eleti, A., Eloundou, T., Farhi, D., Fedus,

L., Felix, N., Fishman, S. P., Forte, J., Fulford, I., Gao, L., Georges, E., Gibson, C., Goel, V., Gogineni, T., Goh, G., Gontijo-Lopes, R., Gordon, J., Grafstein, M., Gray, S., Greene, R., Gross, J., Gu, S. S., Guo, Y., Hallacy, C., Han, J., Harris, J., He, Y., Heaton, M., Heidecke, J., Hesse, C., Hickey, A., Hickey, W., Hoeschele, P., Houghton, B., Hsu, K., Hu, S., Hu, X., Huizinga, J., Jain, S., Jain, S., Jang, J., Jiang, A., Jiang, R., Jin, H., Jin, D., Jomoto, S., Jonn, B., Jun, H., Kaftan, T., Łukasz Kaiser, Kamali, A., Kanitscheider, I., Keskar, N. S., Khan, T., Kilpatrick, L., Kim, J. W., Kim, C., Kim, Y., Kirchner, J. H., Kiros, J., Knight, M., Kokotajlo, D., Łukasz Kondraciuk, Kondrich, A., Konstantinidis, A., Kosic, K., Krueger, G., Kuo, V., Lampe, M., Lan, I., Lee, T., Leike, J., Leung, J., Levy, D., Li, C. M., Lim, R., Lin, M., Lin, S., Litwin, M., Lopez, T., Lowe, R., Lue, P., Makanju, A., Malfacini, K., Manning, S., Markov, T., Markovski, Y., Martin, B., Mayer, K., Mayne, A., McGrew, B., McKinney, S. M., McLeavey, C., McMillan, P., McNeil, J., Medina, D., Mehta, A., Menick, J., Metz, L., Mishchenko, A., Mishkin, P., Monaco, V., Morikawa, E., Mossing, D., Mu, T., Murati, M., Murk, O., Mély, D., Nair, A., Nakano, R., Nayak, R., Neelakantan, A., Ngo, R., Noh, H., Ouyang, L., O'Keefe, C., Pachocki, J., Paino, A., Palermo, J., Pantuliano, A., Parascandolo, G., Parish, J., Parparita, E., Passos, A., Pavlov, M., Peng, A., Perelman, A., de Avila Belbute Peres, F., Petrov, M., de Oliveira Pinto, H. P., Michael, Pokorny, Pokrass, M., Pong, V. H., Powell, T., Power, A., Power, B., Proehl, E., Puri, R., Radford, A., Rae, J., Ramesh, A., Raymond, C., Real, F., Rimbach, K., Ross, C., Rotsted, B., Roussez, H., Ryder, N., Saltarelli, M., Sanders, T., Santurkar, S., Sastry, G., Schmidt, H., Schnurr, D., Schulman, J., Selsam, D., Sheppard, K., Sherbakov, T., Shieh, J., Shoker, S., Shyam, P., Sidor, S., Sigler, E., Simens, M., Sitkin, J., Slama, K., Sohl, I., Sokolowsky, B., Song, Y., Staudacher, N., Such, F. P., Summers, N., Sutskever, I., Tang, J., Tezak, N., Thompson, M. B., Tillet, P., Tootoonchian, A., Tseng, E., Tuggle, P., Turley, N., Tworek, J., Uribe, J. F. C., Vallone, A., Vijayvergiya, A., Voss, C., Wainwright, C., Wang, J. J., Wang, A., Wang, B., Ward, J., Wei, J., Weinmann, C., Welihinda, A., Welinder, P., Weng, J., Weng, L., Wiethoff, M., Willner, D., Winter, C., Wolrich, S., Wong, H., Workman, L., Wu, S., Wu, J., Wu, M., Xiao, K., Xu, T., Yoo, S., Yu, K., Yuan, Q., Zaremba, W., Zellers, R., Zhang, C., Zhang, M., Zhao, S., Zheng, T., Zhuang, J., Zhuk, W., and Zoph, B. (2024). GPT-4 technical report. `https://arxiv.org/abs/2303.08774`.

Opiełka, G., Rosenbusch, H., Vijverberg, V., and Stevenson, C. E. (2024). Do large language models solve ARC visual analogies like people do? `https://arxiv.org/abs/2403.09734`.

Ouellette, S. (2024). Towards efficient neurally-guided program induction for ARC-AGI. `https://arxiv.org/abs/2411.17708`.

Ouellette, S., Pfister, R., and Jud, H. (2024). Counting and algorithmic generalization with transformers. `https://arxiv.org/abs/2310.08661`.

Plate, T. (1991). Holographic reduced representations: Convolution algebra for compositional distributed representations. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'91, page 30–35, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Plate, T. A. (1992). Holographic recurrent networks. In Hanson, S., Cowan, J., and Giles, C., editors, *Advances in Neural Information Processing Systems*, volume 5. Morgan-Kaufmann.

Plate, T. A. (1995). Holographic reduced representations. *IEEE Transactions on Neural Networks*, 6(3):623–641.

Plate, T. A. (2003). *Holographic Reduced Representation: Distributed Representation for Cognitive Structures*. CSLI Publications.

Puget, J.-F. (2024). A 2D nGPT model for ARC Prize. `https://github.com/jfpuget/ARC-AGI-Challenge-2024/blob/main/arc.pdf`.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners.

Rasmussen, D. (2010). A neural modelling approach to investigating general intelligence. Master's thesis, University of Waterloo.

Rasmussen, D. and Eliasmith, C. (2011). A neural model of rule generation in inductive reasoning. *Topics in Cognitive Science*, 3:140–153.

Raven, J. (1936). *The Performances of Related Individuals in Tests Mainly Educative and Mainly Reproductive Mental Tests Used in Genetic Studies*. University of London (King's College).

Rocha, F. M., Dutra, I., Costa, V. S., and Reis, L. P. (2025). Program synthesis using inductive logic programming for the Abstraction and Reasoning Corpus. In Bruno, P., Calimeri, F., Cauteruccio, F., and Terracina, G., editors, *Hybrid Models for Coupling Deductive and Inductive Reasoning*, pages 52–68. Springer Nature Switzerland.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986a). Learning representations by back-propagating errors. *Nature*, 323:533–536.

Rumelhart, D. E., McClelland, J. L., Group, P. R., et al. (1986b). *Parallel distributed processing, volume 1: Explorations in the microstructure of cognition: Foundations*. MIT Press.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117.

Shojaee, P., Mirzadeh, I., Alizadeh, K., Horton, M., Bengio, S., and Farajtabar, M. (2025). The illusion of thinking: Understanding the strengths and limitations of reasoning models via the lens of problem complexity. `https://arxiv.org/abs/2506.06941`.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Singh, M., Cambronero, J., Gulwani, S., Le, V., and Verbruggen, G. (2023). Assessing GPT4-V on structured reasoning tasks. `https://arxiv.org/abs/2312.11524`.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2014). Intriguing properties of neural networks. `https://arxiv.org/abs/1312.6199`.

Thoms, L. H., Veldkamp, K. A., Rosenbusch, H., and Stevenson, C. E. (2023). Solving ARC visual analogies with neural embeddings and vector arithmetic: A generalized method. `https://arxiv.org/abs/2311.08083`.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Voelker, A. R. (2020). A short letter on the dot product between rotated Fourier transforms. `https://arxiv.org/abs/2007.13462`.

Wang, G., Li, J., Sun, Y., Chen, X., Liu, C., Wu, Y., Lu, M., Song, S., and Yadkori, Y. A. (2025). Hierarchical reasoning model. `https://arxiv.org/abs/2506.21734`.

Wang, R., Zelikman, E., Poesia, G., Pu, Y., Haber, N., and Goodman, N. (2024). Hypothesis search: Inductive reasoning with language models. In *The Twelfth International Conference on Learning Representations*.

Wason, P. C. and Evans, J. S. B. T. (1974). Dual processes in reasoning? *Cognition*, 3(2):141–154.

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., Chi, E. H., Hashimoto, T., Vinyals, O., Liang, P., Dean, J., and Fedus, W. (2022). Emergent abilities of large language models. *Transactions on Machine Learning Research*.

Weizenbaum, J. (1966). ELIZA—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45.

Wind, J. S. (2020). 1st place solution + code and official documentation. `https://www.kaggle.com/competitions/abstraction-and-reasoning-challenge/writeups/icecuber-1st-place-solution-code-and-official-docu`.

Witt, J., Dumančić, S., Guns, T., and Carbon, C.-C. (2025). A divide, align, and conquer strategy for program synthesis. *Journal of Artificial Intelligence Research*, 82:1961–1997.

Wu, Z., Qiu, L., Ross, A., Akyürek, E., Chen, B., Wang, B., Kim, N., Andreas, J., and Kim, Y. (2024). Reasoning or reciting? Exploring the capabilities and limitations of language models through counterfactual tasks. In Duh, K., Gomez, H., and Bethard, S., editors, *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1819–1862. Association for Computational Linguistics.

Xu, Y., Khalil, E. B., and Sanner, S. (2023). Graphs, constraints, and search for the Abstraction and Reasoning Corpus. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'23/IAAI'23/EAAI'23. AAAI Press.

Xu, Y., Li, W., Vaezipoor, P., Sanner, S., and Khalil, E. B. (2024). LLMs and the Abstraction and Reasoning Corpus: Successes, failures, and the importance of object-based representations. *Transactions on Machine Learning Research*.

Yehudai, G., Kaplan, H., Ghandeharioun, A., Geva, M., and Globerson, A. (2024). When can transformers count to n? *CoRR*.

Yuan, Z., Yuan, H., Tan, C., Wang, W., and Huang, S. (2023). How well do large language models perform in arithmetic tasks? `https://arxiv.org/abs/2304.02015`.

Zhang, X. and Sheng, V. S. (2024). Neuro-symbolic AI: Explainability, challenges, and future trends. `https://arxiv.org/abs/2411.04383`.