Essay

Teaching Computers to Think Like Us: Cracking the Code of Visual Puzzles

Robert Friedman *

Posted Date: 21 May 2025

doi: 10.20944/preprints202505.1677.v1

Keywords: Abstraction and Reasoning Corpus; ARC; Atomic Visual Primitives; Domain-Specific Language; DSL; Program Synthesis

 doi:10.20944/preprints202505.1677.v1

Essay

Teaching Computers to Think Like Us: Cracking the Code of Visual Puzzles

Robert Friedman †

Department of Biological Sciences, University of South Carolina, Columbia, SC 29208, USA; bob.friedman.2@gmail.com

† Retired.

Abstract: Imagine a puzzle, not of jigsaw pieces, but of abstract visual patterns. You see a few examples of how a grid of colored squares changes – perhaps shapes move, colors shift, or objects duplicate. Your task, after seeing just a handful of these "before and after" snapshots, is to figure out the underlying rule and apply it to a new, unseen grid. This is the essence of the Abstraction and Reasoning Corpus (ARC), a set of challenges designed to push artificial intelligence beyond mere pattern matching into the realm of genuine, human-like understanding. Solving ARC isn't just about smarter algorithms; it's about teaching computers to think in a way that mirrors our own remarkable ability to learn, adapt, and see the hidden logic in the world around us.

Keywords: abstraction and reasoning corpus; ARC; atomic visual primitives; domain-specific language; DSL; program synthesis

1. Introduction

Imagine a puzzle, not of jigsaw pieces, but of abstract visual patterns. You see a few examples of how a grid of colored squares changes – perhaps shapes move, colors shift, or objects duplicate. Your task, after seeing just a handful of these "before and after" snapshots, is to figure out the underlying rule and apply it to a new, unseen grid. This is the essence of the Abstraction and Reasoning Corpus (ARC), a set of challenges designed to push artificial intelligence beyond mere pattern matching into the realm of genuine, human-like understanding. Solving ARC isn't just about smarter algorithms; it's about teaching computers to think in a way that mirrors our own remarkable ability to learn, adapt, and see the hidden logic in the world around us.

Our approach to tackling these fascinating puzzles is inspired by how humans learn and how we build complex systems. Think about how a child learns a language. They don't start with grammar books. They listen, they observe, they associate words with objects and actions ("ball," "roll"), and gradually, they begin to string these "word-primitives" together to form meaningful sentences. Similarly, we envision an AI that learns its own "visual language." It starts by identifying the most basic ways a grid can change – simple "atomic" actions like changing a single square's color, moving a small block, or copying a tiny pattern. These are the fundamental "letters" of its visual alphabet.

Once the AI has a grasp of these basic building blocks, the real learning begins. Just as words combine to form phrases and sentences with richer meaning, our AI learns to combine its atomic visual actions into more complex, multi-step operations, or "visual words" and "phrases." For example, it might learn that a sequence of "identify largest red shape," "find its center," and "mirror all its pixels across that center" is a common and useful operation – effectively learning the concept of "reflect the largest red shape." This learning happens by observing many puzzle solutions, much like a student learns by studying worked examples. The AI uses advanced techniques, akin to those that power modern language translation and understanding tools (like Transformer networks), to spot these recurring, useful combinations and understand their "meaning" – that is, the visual outcome they reliably produce. Importantly, this understanding isn't just abstract; it's constantly

 doi:10.20944/preprints202505.1677.v1

checked against the actual visual changes on the grid, ensuring the AI's internal "language" is firmly grounded in what it "sees."

Over time, the AI builds up an entire library of these learned visual operations, from simple to quite sophisticated. This library becomes its internal toolkit, a Domain-Specific Language (DSL) designed specifically for solving visual puzzles. When faced with a new ARC challenge, the AI doesn't just randomly try things. It acts more like an experienced puzzle-solver or even a software developer. First, it tries to see if the puzzle can be solved using one of its powerful, pre-learned "library functions" – these are the complex visual "phrases" it has mastered. This is like quickly recognizing a familiar type of problem. If a quick solution isn't obvious, the AI doesn't give up. It "digs deeper," breaking the problem down and trying to construct a solution by combining its more basic "atomic" actions, much like a developer writing new code from fundamental commands. This search for a solution is intelligent, guided by what it has learned about which combinations are likely to work, and it constantly refers back to the few puzzle examples it was given to ensure its emerging "program" of actions correctly explains all of them.

This layered approach – from basic visual perception to learning atomic actions, then composing them into a rich visual language, and finally using that language to reason and construct solutions – mirrors the elegance and power of many successful complex systems. Think of 3D graphics in movies and games: artists don't place every pixel; they use software built on basic primitives (like points and polygons) that are composed into functions to draw spheres, apply textures, and simulate light, ultimately creating breathtaking scenes. Our AI, in its own way, learns to become a "developer" of visual logic, building solutions from its learned repertoire.

2. Delving into How the AI Learns and Solves

So, how does this AI actually learn to "see" and "reason" about these visual puzzles? It's a step-by-step process, much like building a house from the foundation up.

First, we equip the AI with a basic "sense of sight" – a digital toolkit that allows it to process the raw grid of colored squares. This isn't about understanding the meaning yet, just about perceiving the fundamental components. It learns to identify distinct objects or shapes based on connected squares of the same color. For each shape it "sees," it notes down its properties: what color is it? How big is it? Where is it on the grid? What other shapes is it next to? This initial processing turns a confusing jumble of pixels into a more organized list of distinct visual elements and their characteristics, much like our eyes and early visual brain regions quickly parse a scene into basic forms and colors before we consciously recognize specific objects.

Table 1. Layer 1 - Low-Level Visual Processing (Perceptual Frontend). This table outlines the initial stage where the AI system processes the raw visual input from ARC puzzle grids into a structured, feature-rich representation of "proto-objects.".

Component	Description	Example Output for a Proto-Object
Core Function	Processes raw ARC grid into structured object/feature representation.	[ObjectID: "obj_001", Color: 2 (Blue), BBox: (x:1,y:1,w:3,h:2), Centroid: (2.5, 2.0), ...]
Input	Raw ARC grid (2D array of color values).	-
Processing Steps	1. Identify connected components of the same color.	-
	2. Treat each component as a "proto-object."	-
	3. Extract features for each proto-object.	-

 doi:10.20944/preprints202505.1677.v1

Output Features *(per proto-object)*	- ObjectID (unique identifier)	"obj_001"
	- Color (integer value)	2 (representing Blue)
	- BBox (bounding box: x, y, width, height)	(1, 1, 3, 2)
	- Centroid (geometric center: x_c, y_c)	(2.5, 2.0)
	- PixelCount (integer)	6
	- ShapeSignature (e.g., Hu moments, aspect ratio, convexity)	Vector of Hu moments, AspectRatio: 1.5
	- AdjacencyList (list of ObjectIDs of neighboring proto-objects and their contact points/edges)	["obj_002_right_edge", "obj_003_bottom_edge"]
Overall Output	A list of structured proto-object descriptions for each input and output grid in an ARC example pair.	List of [ObjectID, Color, BBox, Centroid, PixelCount, ShapeSignature, AdjacencyList]

With this basic perception in place, the AI embarks on discovering its "atomic visual actions" – the simplest, indivisible ways the grid can change. Imagine giving the AI a set of very rudimentary tools: one that can change a pixel's color, another that can shift a small region of pixels, another that can rotate a tiny pattern, and so on. The AI then experiments by trying out short sequences of these tool actions on the "before" grids from various puzzle examples. If a sequence of actions successfully transforms a "before" grid into its corresponding "after" grid, that's a promising sign. But the AI is a discerning learner; it's looking for the most fundamental actions. So, it checks if an even shorter part of that successful sequence could have achieved the same result. If it finds a sequence that works and can't be simplified any further for that specific example, that sequence is a candidate for an "atomic action."

Table 2. Layer 2 - Discovery and Definition of Atomic Visual Primitives. This table describes the process by which the AI system identifies or learns a foundational set of elementary, non-decomposable visual operations that form its basic "alphabet" of change.

Component	Description	Example
Core Function	Identifies/learns foundational, non-decomposable visual operations ("atomic primitives").	Learns "TranslateObjectBySmallDelta" or "SetRegionToSingleColor".
1. Define Basic Operations	Initial set of extremely low-level, parameterized grid manipulation functions.	PixelSetColor(coords, color), RegionTranslate(pixel_list, dx, dy), RegionRotate(pixel_list, pivot, angle_90_deg_multiples).
2. Generate Candidate Primitives	Systematically or stochastically combine 1-3 basic operations. Applied to proto-objects (from Layer 1) or regions derived from them. Prioritize operations on same/related objects.	[RegionRotate(obj1.pixels, obj1.centroid, 90); RegionTranslate(obj1.pixels, 2, 0)] applied to a specific obj1.

 doi:10.20944/preprints202505.1677.v1

3. Test Candidates	Apply candidate sequences to input proto-objects/grids from ARC examples. Check for precise pixel-level or object-property-level match with the corresponding output example.	Candidate C transforms input grid I to I'. Compare I' with target output grid O.
4. Non-Decomposability Filter	For a successful candidate C = [op1, op2, op3], check if any strict sub-sequence (e.g., [op1, op2]) also achieves the same specific transformation I -> O. Retain C if no simpler version works.	If [Rotate; Translate] works, but just Translate also (coincidentally) works for *this specific I,O pair*, then [Rotate; Translate] is not considered atomic *for this instance*. Goal: minimal effective sequence for the observed effect.
5. Cluster & Define Atomic Primitives		
A. Representation for Clustering	Successful, non-decomposable candidate sequences are represented by a feature vector describing their *effect* on the input.	Effect vector: (delta_x_centroid, delta_y_centroid, delta_rotation_signature, color_change_distribution_vector, size_change_ratio, ...)
B. Clustering Algorithm	E.g., DBSCAN, k-means, or spectral clustering on the effect vectors.	DBSCAN groups vectors of similar transformation effects.
C. Atomic Primitive Definition	Each robust cluster centroid (or a representative sequence from the cluster) is defined as a parameterized "atomic primitive" type. This forms the vocabulary for Layer 3.	A cluster might represent "TranslateObjectRightByOne Unit(object_query)". Example: FillSquare(size=2, color=BLUE, position_query)

To make sense of all these candidates, the AI groups them. For instance, many different tiny pixel-moving sequences might all achieve the same effect – say, "move a small object one step to the right." These similar-effect sequences are clustered together, and the core idea of that cluster (like "MoveRightSmall") becomes one of the AI's defined atomic actions. This is how it builds its foundational vocabulary of visual change.

Once the AI has this vocabulary of atomic actions, it moves on to the truly exciting part: learning to combine them into more complex and meaningful "visual sentences" or "compositions." This is where powerful AI techniques like Transformers come into play – the same kind of AI that helps understand and generate human language. We feed the AI a "corpus" or library of solved puzzles, where each solution is written down as a sequence of the atomic actions it learned earlier. For example, a solution might look like: "Take the red square, move it three steps right; then, change the red square's color to blue." The Transformer AI reads through countless such solution sequences. It learns which atomic actions often appear together, which ones typically follow others, and what kinds of visual transformations these combinations achieve.

To make sure this learning isn't just abstract symbol shuffling, we constantly ground it in visual reality. The AI isn't just looking at sequences of action names; it's also "shown" the "before" grid, the "after" grid, and perhaps even a representation of the difference between them. It learns to connect a sequence of actions like "IdentifyLargestObject; MoveToCenter" not just as a string of symbols, but with the actual visual outcome of the largest object in the grid moving to the middle. This connection between its internal "language" and what it "sees" is crucial for genuine understanding.

 doi:10.20944/preprints202505.1677.v1

Table 3. Layer 3 - Learning Compositions & Codifying into a Domain-Specific Language (DSL). This table details how the AI learns to combine atomic primitives into more complex, reusable transformations ("compositions" or "visual words/phrases") and formalizes them into an internal Domain-Specific Language (DSL).

Component	Description	Example
Core Function	Learns how atomic primitives (from Layer 2) compose into reusable, complex transformations. Codifies these compositions as parameterized functions in an internal DSL.	Learns a composition like "ReflectObjectAndChangeColor" and defines it as ReflectAndRecolor(obj_query, axis, new_color) in the DSL.
1. Corpus Generation	Solutions to ARC training puzzles (or sub-problems) are represented as sequences of parameterized atomic primitives.	[Translate(obj_id=ID_RedSquare, dx=3, dy=0); Recolor(obj_id=ID_RedSquare, target_color=BLUE)]
2. Transformer-based Semantic Learning	A Transformer model is trained on this corpus of primitive sequences.	-
A. Input Tokens to Transformer	Abstracted sequences: [CLS] Translate OBJ_TYPE_SQUARE COLOR_RED dx_3 dy_0 [SEP] Recolor OBJ_TYPE_SQUARE COLOR_PREV new_color_BLUE [SEP] ...	-
B. Training Objectives	Self-supervised tasks like masked primitive modeling (predicting missing primitives in a sequence) or next primitive prediction.	-
3. Multimodal Grounding	Ensures learned semantics of primitive sequences are linked to their visual effects.	-
A. Augmented Transformer Input	(tokenized_primitive_sequence, input_grid_visual_features, output_grid_visual_features)	-
B. "Visual Delta" Representation	A feature vector representing the change between input and output grids (e.g., from a Siamese network comparing grid embeddings, or a structured list of object property changes: [(obj_id, prop, old, new), ...]).	-
C. Joint Training Objective	E.g., Loss = Loss_sequence_prediction + Loss_visual_delta_prediction + Loss_output_grid_reconstruction_from_state.	-
4. DSL Codification	Frequent, high-utility, and semantically coherent sequences (compositions) identified from Transformer analysis (e.g., attention patterns, generation probabilities) or statistical analysis of solution traces are formalized into the DSL.	-

 doi:10.20944/preprints202505.1677.v1

A. Identifying Compositions	Look for sub-sequences that reliably achieve a recognizable sub-goal or transformation type across diverse puzzles.	-
B. Formalization & Parameterization	Abstract the specific entities/values in a common sequence to create parameterized functions. A human or automated system inspects sequences.	Sequence [IdentifyLargestObject(color=ANY); MoveToPosition(obj_id=LARGEST, pos=CENTER)] becomes DSL function DEFINE Function CenterLargestObject().
C. Example DSL Function	```dsl Function AlignObjectsVertically(obj_list_query, spacing_y): obj_list = QueryObjects(grid, obj_list_query) // Uses Layer 1 output SortObjectsByY(obj_list) current_y = obj_list [0].y FOR obj IN obj_list: MoveToY(obj.id, current_y) // Uses an atomic primitive/DSL func current_y = current_y + obj.height + spacing_y ENDFOR EndFunction ```	

When the AI consistently finds a sequence of atomic actions that is useful and solves a common type of visual sub-problem, it "codifies" this sequence. It essentially gives it a name and turns it into a new, more powerful function in its internal toolkit – its own Domain-Specific Language (DSL). For instance, a complex series of pixel manipulations that always results in mirroring an object might become a single function called ReflectObject(object_to_reflect, along_which_axis).

Finally, equipped with its perception skills, its vocabulary of atomic actions, and its library of powerful composed functions (its DSL), the AI is ready to tackle new, unseen ARC puzzles. When a new puzzle arrives, the AI first tries to solve it using its high-level "library functions." It's like an expert quickly checking if the problem fits a known pattern that has a standard solution method. For example, it might try its ReflectObject function with different parameters, or its AlignObjectsVertically function. It cleverly figures out the right parameters (like which object to reflect or how much spacing for alignment) by looking at the few "before and after" examples provided with the puzzle. If these examples show a red square moving three steps right, the AI infers that the "move distance" parameter should be "three steps right."

If none of its high-level functions provide a quick solution within a set time, the AI doesn't give up. It shifts gears and tries a more detailed approach, attempting to construct a solution from its more basic atomic actions. This is a more painstaking search, but it allows the AI to tackle truly novel problems that don't neatly fit its pre-learned complex operations. Throughout this whole process, whether it's trying high-level functions or basic actions, the AI constantly checks its proposed sequence of operations against all the provided puzzle examples. Any "program" it devises must

 doi:10.20944/preprints202505.1677.v1

correctly transform every single "before" grid into its corresponding "after" grid to be considered a valid solution.

Table 4. Layer 4 - Formal System for Solution Synthesis (The "ARC-Solver 2.0" Code). This table outlines the final stage where the AI utilizes its learned Domain-Specific Language (DSL), comprising atomic primitives and codified compositions, to search for and construct solutions ("programs") for new ARC puzzles.

Component	Description	Example / Elaboration
Core Function	Utilizes the DSL to search for and construct programs (sequences of DSL function calls) that solve new ARC puzzles, based on few-shot examples.	Given a new puzzle, synthesizes a program like [CenterLargestObjectOfColor(RED); DuplicateAndOffset(LAST_RESULT, dx=5, dy=0)].
1. Problem Representation	Input/output examples of a new ARC puzzle are parsed by Layer 1 into structured proto-object representations.	-
2. Time-Budgeted Hierarchical Search	A two-phase search strategy.	-
A. Phase A (High-Level Search)	System first attempts to solve the puzzle by applying parameterized DSL functions (learned compositions from Layer 3). Search space is over high-level operations.	Tries functions like AlignObjectsVertically, ReflectObjectGroup, etc., inferring parameters.
B. Phase B (Low-Level Search/Refinement)	If Phase A fails or exceeds time budget, system falls back to searching with atomic primitives (from Layer 2) or tries to refine/combine partial solutions from Phase A. Search space is more granular.	Attempts to build solution step-by-step using Move, Recolor, Rotate atomic primitives.
3. Solution Synthesis Engine	Core reasoning module for constructing solution programs.	-
A. Program Synthesis Approach	Search-based program synthesis: finding a sequence of DSL function calls that transforms all input examples to their respective outputs.	E.g., Beam search or A* search where states are (current_grid_configuration, partial_program).
B. Guided Search Heuristic	Search algorithm (e.g., MCTS, A*) can be guided by a policy network (potentially derived from the Layer 3 Transformer) that predicts promising next DSL functions or atomic primitives to apply.	Transformer predicts `P(next_DSL_function
C. Parameter Inference	For a selected DSL function, infer its parameters based on the few-shot examples.	If Move(obj_id, dx, dy) is selected: observe obj_A moves from (x1,y1) to (x2,y2) in example 1. Candidate params: dx=x2-x1, dy=y2-y1. Verify

 doi:10.20944/preprints202505.1677.v1

		consistency across all examples.
D. Few-Shot Example Pruning	A candidate program (sequence of DSL functions with inferred parameters) is only valid if it correctly transforms *all* provided input-output examples. Incorrect programs are pruned from the search.	If program P works for example 1 but fails on example 2, P is discarded.
Overall Output	A program (a sequence of parameterized DSL functions) that solves the given ARC puzzle, or failure if no solution is found within budget.	SolvePuzzle() -> [DSL_FunctionCall1(params), DSL_FunctionCall2(params), ...].

This careful, multi-layered approach – perceiving basic elements, learning atomic actions, composing them into a rich visual language, and then using this language to intelligently search for solutions – is how we aim to build an AI that can truly master the art of abstract visual reasoning. We might even start its "education" with simpler puzzles, gradually increasing the difficulty, much like a human student learns, to make the process more manageable.

3. What We Expect and Other Paths to Explore

What do we hope to achieve with this sophisticated learning system? Our primary goal is, of course, to create an AI that can successfully solve a wide range of ARC puzzles, especially those that require putting together multiple logical steps in creative ways. We expect it to demonstrate "compositional generalization" – the very human-like ability to take learned building blocks (its atomic actions and simpler compositions) and combine them in new ways to solve problems it has never encountered before. As it learns, its internal "visual language" will grow richer and more expressive, capturing a deeper understanding of the kinds of transformations possible in the ARC world. We even anticipate that it might discover clever visual "idioms" or strategies that are effective but not immediately obvious to human observers.

Our confidence in this approach is substantial. The core technologies we plan to use – like Transformer networks for learning from sequences and well-established search algorithms – have proven their power in many other challenging AI domains. The idea of building understanding in layers, from simple perception to complex reasoning, is a principle that works both in human intelligence and in well-designed complex software. And crucially, by ensuring that the AI's internal "language" is always connected to actual visual changes, we tackle a fundamental challenge in AI: ensuring symbols have real meaning. The incremental nature of building this system, where successes at simpler compositions build confidence and data for learning more complex ones, also makes development more manageable. While ARC is designed to be incredibly tough, we believe this structured, learning-focused path offers a very promising direction.

Of course, this isn't the only way one might try to build an ARC-solving AI. Some researchers might attempt to train one enormous, all-encompassing AI brain (a giant neural network) to learn directly from the "before" and "after" grids without explicitly defining these intermediate layers of understanding. While appealing in its simplicity, this "end-to-end" approach often requires colossal amounts of data and can struggle with the kind of precise, step-by-step logical reasoning that ARC demands. Others might lean heavily on traditional symbolic AI, pre-programming the AI with a vast set of logical rules. This can be very powerful for problems with clear rules, but it can be brittle when faced with novelty and often struggles to learn new fundamental concepts from raw visual data. Purely evolutionary approaches, which try to "evolve" solutions, can also be incredibly slow and get

 doi:10.20944/preprints202505.1677.v1

stuck without a lot of guidance. Even the latest large language models, while incredibly versatile, may not yet possess the fine-grained visual manipulation skills and systematic generalization needed for the full range of ARC puzzles without significant specialized adaptation; though hybrid approaches using LLMs to propose high-level strategies for our DSL could be a fruitful avenue. Our chosen path, a hybrid that blends learning, symbolic representation, and intelligent search, aims to capture the best of these worlds, creating an AI that is both knowledgeable and adaptable.

The quest to solve ARC is more than an academic exercise; it's a journey towards understanding the very nature of intelligence, abstraction, and learning. By striving to build systems that can master these puzzles, we are pushing the boundaries of what AI can achieve, bringing us closer to machines that can reason, adapt, and understand the world with a depth and flexibility that begins to approach our own.

Funding: This research received no external funding.

Acknowledgement: The written text along with many of the concepts are derived from consultation with Gemini 2.5 Pro Preview, a model of artificial intelligence by Google (version 05/06/2025).

Conflicts of Interest: The author declares no conflict of interest.

Glossary of Technical Terms

Abstraction and Reasoning Corpus (ARC): A benchmark dataset of visual reasoning puzzles designed to test abstract reasoning and generalization in AI.

Atomic Visual Primitives: The most basic, non-decomposable operations that can alter a visual grid or its components, forming the fundamental vocabulary of the system.

Bounding Box: The smallest rectangle enclosing an object or a region of interest in an image or grid.

Centroid: The geometric center of an object or shape.

Clustering (e.g., k-means, DBSCAN): Unsupervised machine learning techniques for grouping similar data points together based on their features.

Compositionality: The principle that complex representations or processes can be built by combining simpler components in a structured way.

Connected Components: In image analysis, sets of pixels that are connected to each other (e.g., forming a contiguous object of the same color).

Corpus: A large and structured set of texts or, in this context, sequences of operations or puzzle solutions used for training machine learning models.

Curriculum Learning: A training strategy where a model is first trained on easier examples and gradually exposed to more complex ones.

Domain-Specific Language (DSL): A computer language specialized for a particular application domain, in this case, for describing visual transformations in ARC puzzles.

Few-Shot Generalization/Learning: The ability of an AI system to learn a new concept or solve a new task from very few (e.g., 1-5) examples.

Grounding (Symbol Grounding): The problem of connecting symbolic representations (like words or DSL functions) to their meaning in the real or perceptual world.

Heuristic (Search Heuristic): A rule of thumb or an educated guess used to guide a search algorithm towards a solution more efficiently.

Hierarchical Abstraction: A system organization where complex entities are understood as compositions of simpler, lower-level entities, forming multiple layers of abstraction.

Hu Moments: A set of seven image moments that are invariant to translation, scale, and rotation, often used for basic shape description.

Inductive Biases: Assumptions or preferences incorporated into a learning algorithm that guide it towards certain types of solutions or representations.

 doi:10.20944/preprints202505.1677.v1

Monte Carlo Tree Search (MCTS): A heuristic search algorithm often used in game playing and planning, which explores the search space by building a tree based on random sampling and simulated outcomes.

Multimodal Learning: An approach in AI where models learn from and integrate information from multiple types of data (modalities), such as text, images, and structured data.

N-gram: A contiguous sequence of n items from a given sample of text or speech (or in this case, a sequence of primitives).

Parameter Inference: The process of determining the specific values for the parameters (arguments) of a function or model based on observed data.

Policy Network: In reinforcement learning or guided search, a neural network that outputs a probability distribution over possible actions or next steps.

Program Synthesis: The automatic generation of computer programs from a high-level specification or examples.

Proto-object: An initial, basic segmentation of a visual scene into distinct entities before more complex object recognition or understanding.

Self-Supervised Learning: A type of machine learning where a model learns from unlabeled data by creating supervisory signals from the data itself (e.g., predicting a masked part of the input).

Siamese Network: A neural network architecture that uses two or more identical subnetworks to process different inputs and compare their resulting embeddings, often used for similarity learning.

Transformer: A neural network architecture based on self-attention mechanisms, highly effective for processing sequential data like text or sequences of operations.

References

Further Readings and Resources

The approach outlined in this report draws upon a rich tapestry of research in artificial intelligence, cognitive science, and computer science. For readers interested in delving deeper into the foundational concepts and related advancements, the following resources provide excellent starting points:

The Abstraction and Reasoning Corpus (ARC)

1. The ARC Prize Official Website: https://arcprize.org/
   Relevance: The primary source for information about the ARC challenge, datasets, rules, ongoing competitions, and community discussions.
2. Chollet, F. (2019). On the Measure of Intelligence. arXiv:1911.01547.
   Relevance: This paper by the creator of ARC lays out the philosophical and practical motivations behind the dataset, emphasizing the need for AI systems that possess robust abstraction, reasoning, and generalization capabilities, particularly in few-shot learning scenarios.

Core AI Concepts and Architectures

Transformers and Attention Mechanisms:

3. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.
   The seminal paper introducing the Transformer architecture, which is foundational for learning from sequential data (like our proposed sequences of visual primitives) and underpins many modern AI advancements.

Program Synthesis:

4. Gulwani, S., Polozov, O., & Singh, R. (2017). Program synthesis. Foundations and Trends in Programming Languages, 4 (1-2), 1-119.
   A comprehensive overview of program synthesis techniques, which are central to the idea of our AI constructing "programs" (sequences of DSL functions) to solve ARC puzzles.

Object-Centric Learning:

5. Locatello, F., et al. (2020). Object-centric learning with slot attention. Advances in Neural Information Processing Systems, 33, 11525-11535.

 doi:10.20944/preprints202505.1677.v1

Introduces Slot Attention, a key method for learning to parse scenes into discrete object representations without explicit supervision, relevant to our Layer 1 perceptual frontend.

Reinforcement Learning for Algorithmic Discovery:

6. Fawzi, A., Balog, M., Huang, A., Hubert, T., Romera-Paredes, B., Barekatain, M., ... & Schrittwieser, J. (2022). Discovering faster matrix multiplication algorithms with AlphaTensor. Nature, 610 (7930), 47-53.
Demonstrates how deep reinforcement learning can discover novel and efficient algorithms (composed of primitives), analogous to how our system might discover effective compositions of visual primitives.

Cognitive Science and Philosophical Foundations

Symbol Grounding Problem:

7. Harnad, S. (1990). The symbol grounding problem. Physica D: Nonlinear Phenomena, 42 (1-3), 335-346.
A classic paper discussing how symbolic representations in AI can acquire intrinsic meaning through connection to perceptual experiences, directly relevant to our emphasis on grounding the "visual language."

Compositionality in Cognition:

8. Fodor, J. A., & Pylyshyn, Z. W. (1988). Connectionism and cognitive architecture: A critical analysis. Cognition, 28 (1-2), 3-71.
Argues for the necessity of compositional representations in explaining systematicity in human thought, a principle central to our layered, compositional approach.

Neuro-Symbolic AI

9. Garcez, A. D'Avila., & Lamb, L. C. (2020). Neurosymbolic AI: The 3rd Wave. arXiv:2012.05876.
Provides an overview of neuro-symbolic approaches, which aim to combine the strengths of neural networks (learning, perception) with symbolic reasoning (logic, abstraction), aligning with the hybrid nature of the proposed ARC solver.