- Object-Centric Perception: *Slot Attention*

Slot Attention is a neural module designed to extract object-centric representations from perceptual inputs like images or grids, without requiring supervision. It takes feature maps (for example, from a CNN encoder processing a grid) and learns a set of "slots," where each slot competes to bind to different parts of the input through an iterative attention mechanism. During each iteration, the slots compete to explain pieces of the input features, essentially deciding "who owns what" through a soft attention mechanism. As this process repeats, each slot becomes more specialized, binding to specific objects or regions in a scene. This way, the model learns to separate and represent different objects naturally, without explicit supervision or predefined rules. This results in each slot capturing a distinct object or region in the scene.

The slots start from random initial values and, through multiple refinement steps, specialize in representing consistent parts (e.g., individual shapes or color blobs in a grid). Because of its design, Slot Attention naturally handles varying numbers and configurations of objects, making it ideal for tasks like ARC AGI 2 that require flexible object segmentation.

In summary, Slot Attention solves the crucial first step: transforming a raw grid into a set of explicit, interpretable object-level embeddings, which can then be used for higher-level reasoning or transformations in later stages.

Paper: https://arxiv.org/abs/2006.15055

- Concept Abstraction & Invariant Feature Learning: Group Equivariant Convolutional Networks (G-CNNs)

Group Equivariant Convolutional Networks (G-CNNs) extend regular CNNs to handle not just translations but also rotations and reflections directly within the network's design. This is particularly valuable for ARC AGI 2 tasks, where many puzzles involve shapes being rotated, mirrored, or shifted in space. Instead of needing to see every possible variant during training, a G-CNN naturally understands these transformations by using mathematical symmetry groups, such as p4 (rotations by 90 degrees) or p4m (rotations plus reflections).

When using G-CNNs in ARC AGI 2, you choose a symmetry group to define which types of transformations the network should be equivariant to. Choosing the group does **not** tell the network which specific transformation to apply in each puzzle — it simply sets the "rules of the world" so the model understands, for example, that a rotated shape is still the same shape. The actual puzzle-specific transformation logic (like "rotate this object and move it

to the corner") is still learned from the demonstration examples during training. By making the feature maps equivariant to these groups, the model can recognize and process objects consistently, no matter how they are oriented or reflected.

After extracting robust equivariant features, these can be combined with object-centric layers (such as Slot Attention) to separate and describe individual objects in the grid. Overall, using G-CNNs equips the agent with a strong inductive bias toward understanding spatial transformations as "first-class citizens," allowing it to learn abstract puzzle-solving principles rather than just memorizing patterns. This makes the agent more sample-efficient, better at generalizing new tasks, and ultimately more human-like in its reasoning.

Paper: https://arxiv.org/abs/1602.07576

- Hypothesis Generation via Meta-Learning: DreamCoder

DreamCoder can be a powerful foundation for solving ARC AGI 2 tasks because it learns to synthesize programs that describe how to transform inputs into outputs, rather than just predicting final grids. In ARC AGI 2, each task is essentially a hidden symbolic transformation—such as rotating objects, mirroring shapes, or selectively removing elements. DreamCoder's core strength lies in writing explicit, interpretable programs to achieve these transformations, and in gradually building up a library of reusable functions as it solves more tasks.

To apply DreamCoder to ARC AGI 2, you would first define a domain-specific language (DSL) with basic grid operations like rotate, mirror, color-change, and object-level filters. For each demonstration pair, the system would search for a short program in this DSL that reproduces the output grid from the input grid. This is done using DreamCoder's "wake" phase, guided by a neural recognition model that proposes likely subroutines or primitive combinations based on previous experience.

Once it finds working programs, DreamCoder enters its "sleep" phases. In the abstraction sleep phase, it analyzes these programs to identify common patterns and abstracts them into new high-level functions (for example, "remove border objects" might be abstracted from many different puzzle solutions). In the dream sleep phase, it trains its neural model to become better at predicting programs for future tasks, using both real solved tasks and synthetic ones it dreams up.

By repeatedly cycling through solving tasks, abstracting common solutions, and improving its predictions, DreamCoder gradually builds a robust library and becomes increasingly capable of handling new ARC puzzles. Ultimately, this approach allows an ARC agent to

learn true symbolic reasoning and compositionality — enabling it to generalize far beyond simple pattern matching and approach human-like puzzle-solving abilities.

Paper: https://www.neurosymbolic.org/papers/EllisWNSMHCST21.pdf

- Self-Explanation & Consistency Checks : CoT

Chain-of-Thought (CoT) prompting is a method where a model is encouraged to generate intermediate reasoning steps rather than directly outputting a final answer. In their paper, Wei et al. showed that by "thinking aloud," large language models dramatically improve on complex tasks involving arithmetic, logic, and symbolic reasoning — all highly relevant to ARC AGI 2 puzzles.

In ARC AGI 2, each puzzle typically involves a series of abstract, multi-step transformations on grid-based objects. Rather than predicting the final transformed grid directly, a CoT-enabled agent would first describe what it sees, such as identifying and segmenting different shapes or patterns within the grid. The model might articulate, for example: "There are two red L-shapes near the center and a blue square at the bottom left."

Next, the agent would explicitly break down the transformation process into a step-by-step reasoning chain. For instance, it might output: "Step 1: Remove all objects touching the borders. Step 2: Move the remaining objects to the bottom row. Step 3: Change their color to green." By spelling out these symbolic or natural language instructions, the agent effectively turns a complex global transformation into a sequence of smaller, more interpretable operations.

Finally, each sub-step is executed one at a time, allowing for intermediate checks and potential corrections. This stepwise approach not only improves the system's ability to generalize to new puzzle configurations but also makes it more interpretable and easier to debug. By adopting Chain-of-Thought prompting, an ARC AGI 2 agent could reason more like a human — composing solutions through explicit, logical steps rather than simply mimicking patterns from training data.

Paper: https://arxiv.org/abs/2201.11903

- Test-Time Adaptation & Self-Refinement : Meta Test-Time Training (MT3)

In ARC-AGI 2, each new puzzle can be thought of as a new "task" with its own hidden transformation rules. The main challenge is that the model only gets a few demonstration pairs and then must generalize to a new input grid. MT3 provides a perfect framework to

handle this challenge because it is designed to let a model adapt its internal parameters **at test time**, using just a few unlabeled samples.

During ARC puzzle solving, the model would first start from a strong general base (meta-learned parameters). When it receives the few demonstration input-output pairs, it can use them to update its internal representation in a self-supervised way. For example, instead of needing explicit labels for every transformation rule, it would rely on augmented versions of the demo grids (e.g., distorted, shifted, or noised) and minimize a self-supervised loss that encourages the model to learn robust, puzzle-specific features.

Once the model finishes these small, puzzle-specific updates, it would then apply its newly adapted parameters to the unseen test input grid, producing the predicted output grid. This fine-tuned version is used only for that puzzle; afterward, the model resets to its base parameters before solving the next task.

By integrating MT3 in this way, the ARC-AGI 2 agent effectively "figures out" the specific transformation logic of each puzzle in a fast, sample-efficient manner, much like a human might adjust their mental model after seeing a few examples. This makes the system more robust to novel and complex puzzles and greatly improves generalization to tasks it has never explicitly seen before.

Paper: https://arxiv.org/abs/2103.16201

Unseen Puzzle (Demo Pairs)

↓

Slot Attention → Object embeddings

↓

G-CNN → Invariant, robust object features

↓

DreamCoder → Synthesized symbolic program

↓

Chain-of-Thought → Break into explicit steps

↓

MT3 → Adapt to puzzle-specific variations

↓

Run program → Final Output Grid