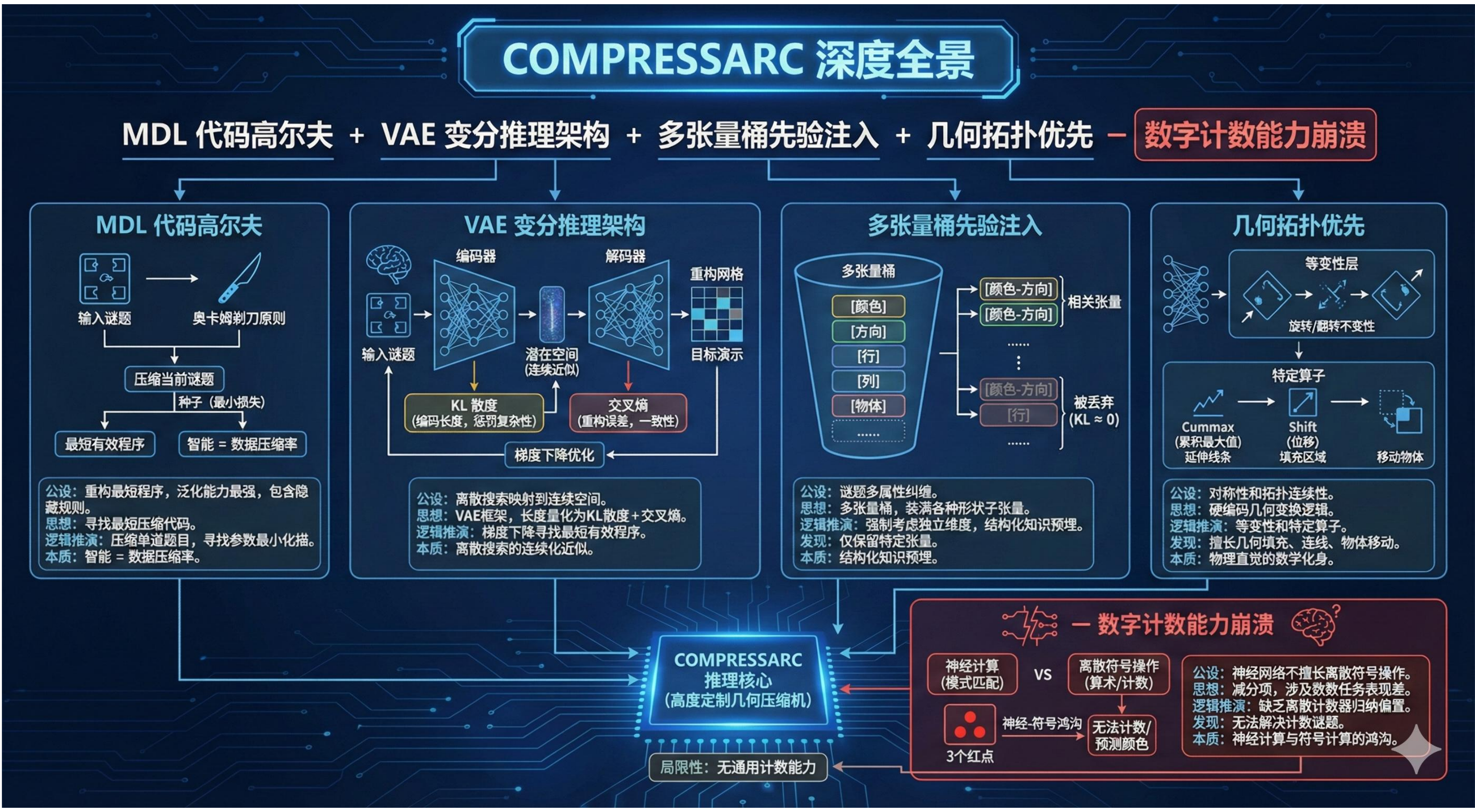


COMPRESSARC 深度全景

MDL 代码高尔夫 + VAE 变分推理架构 + 多张量桶先验注入 + 几何拓扑优先 − 数字计数能力崩溃

MDL 代码高尔夫

输入谜题 → 奥卡姆剃刀原则

压缩当前谜题

种子（最小损失）

最短有效程序

智能 = 数据压缩率

公设：重构最短程序，泛化能力最强，包含隐藏规则。
思想：寻找最短压缩代码。
逻辑推演：压缩单道题目，寻找参数最小化描。
本质：智能 = 数据压缩率。

VAE 变分推理架构

输入谜题 → 编码器 → 潜在空间（连续近似） → 解码器 → 重构网格

目标演示

KL 散度（编码长度，惩罚复杂性）

交叉熵（重构误差，一致性）

梯度下降优化

公设：离散搜索映射到连续空间。
思想：VAE框架，长度量化为KL散度＋交叉熵。
逻辑推演：梯度下降寻找最短有效程序。
本质：离散搜索的连续化近似。

多张量桶先验注入

多张量桶

[颜色]
[方向]
[行]
[列]
[物体]
……

[颜色-方向]
[颜色-方向]
相关张量

……

[颜色-方向]
[行]
被丢弃（KL ≈ 0）

……

公设：谜题多属性纠缠。
思想：多张量桶，装满各种形状子张量。
逻辑推演：强制考虑独立维度，结构化知识预埋。
发现：仅保留特定张量。
本质：结构化知识预埋。

几何拓扑优先

等变性层

旋转/翻转不变性

特定算子

Cummax（累积最大值）延伸线条 → Shift（位移）填充区域 → 移动物体

公设：对称性和拓扑连续性。
思想：硬编码几何变换逻辑。
逻辑推演：等变性和特定算子。
发现：擅长几何填充、连线、物体移动。
本质：物理直觉的数学化身。

COMPRESSARC
推理核心
（高度定制几何压缩机）

局限性：无通用计数能力

− 数字计数能力崩溃

神经计算（模式匹配） VS 离散符号操作（算术/计数）

3个红点 —神经-符号鸿沟→ 无法计数/预测颜色

公设：神经网络不擅长离散符号操作。
思想：减分项，涉及数数任务表现差。
逻辑推演：缺乏离散计数器归纳偏置。
发现：无法解决计数谜题。
本质：神经计算与符号计算的鸿沟。

CompressARC：无需预训练的 ARC-AGI

https://gemini.google.com/share/b046bee107f4
Created with **Fast** 25 January 2026 at 09:58 Published on 25 January 2026 at 12:40

只进行翻译就行，两页两页的进行

这里是文档第 1 页和第 2 页的中文翻译。

第 1 页

无需预训练的 ARC-AGI

摘要 在大语言模型（LLM）时代，传统的观点认为，解决 ARC-AGI-1 基准测试中类似智商测试的视觉谜题需要源自大规模预训练的能力。 为了反驳这一点，我们推出了 CompressARC，这是一个没有任何预训练的 7.6 万参数模型，它仅在推理阶段通过最小化目标谜题的描述长度（MDL）来解决了 20% 的评估谜题。 MDL 赋予了 CompressARC 在深度学习中通常闻所未闻的极强泛化能力。 据我们所知，CompressARC 是目前唯一的 ARC-AGI 深度学习方法，其训练仅发生在一个样本上：即去除了最终解决方案信息的目标推理谜题本身。 此外，CompressARC 并不在预先提供的 ARC-AGI “训练集”上进行训练。 在这种数据极其受限的条件下，通常我们不期望能解决任何谜题。 然而，CompressARC 仍然解决了一系列多样化且富有创意的 ARC-AGI 谜题，这表明 MDL 可能是除了传统预训练之外，另一种产生智能的可行途径。

引言 ARC-AGI-1 基准测试由抽象的视觉推理谜题组成，旨在评估系统从极少量输入数据中快速获取新技能的能力 (Chollet, 2019)。 近期基于 LLM 的推理进展展示了令人印象深刻的技能获取能力，但这些系统仍然依赖于大量的预训练数据。 在本文中，我们通过引入 CompressARC 来探索解决 ARC-AGI 真正需要多少数据，这是一种源自最小描述长度（MDL）原则的求解方法 (Rissanen, 1978)。 CompressARC 所有的学习都在推理时进行，并仅使用当前待解决的谜题作为输入数据，就在 ARC-AGI-1 评估谜题上达到了 20% 的准确率。

CompressARC 极高数据效率的关键在于将其表述为一个“代码高尔夫”(code-golfing）问题：寻找能够输出整个 ARC-AGI 数据集（其中任何未解的网格被任意填充）的最短自包含程序。 根据奥卡姆剃刀原则，我们可以预期最短的程序会以最合

理的方式填充未解网格，即符合谜题规则的“正确”解。 一个过于基础的程序可能会存储谜题数据的硬编码字符串（加上任意解）用于输出；但这样的程序会太长，意味着输出的解将是错误的。 另一方面，寻找最优的最短程序需要穷举许多候选程序，这在计算上是不可行的 (Solomonoff, 1964; Hutter, 2005)。

CompressARC 通过让神经网络过拟合谜题数据以将谜题压缩成权重矩阵，达成了一种新型的平衡；这些权重可以硬编码进程序中以替代谜题本身，然后在程序内部用于恢复记忆的谜题及其解。 通过仔细计算硬编码权重的比特长度，该技术将寻找最佳程序的组合搜索转化为可微优化问题，使我们要能在合理的时间内最小化程序长度，并生成良好的解预测。 这种框架保留了原始代码高尔夫表述的几个吸引人的特性，这些特性在 ARC-AGI 的神经求解方案中都是新颖的： • **无预训练**：由于我们从手头的目标谜题开始，不需要训练阶段。

第 2 页

（图 1 及列表续）

- **推理时学习**：程序长度仅在推理过程中通过针对目标谜题优化网络权重来最小化。
- **最小数据需求**：遵循奥卡姆剃刀原则，我们假设最短程序具有强大的泛化能力，并且仅使用谜题本身——不将任何额外数据加载到内存中。

尽管从未使用训练集、未进行预训练，且网络中只有 7.6 万个参数，CompressARC 仍泛化并解决了 20% 的评估谜题和 34.75% 的训练谜题——在这些限制条件下，这种表现对于传统深度学习方法来说是不可能的。 CompressARC 在此设置下的强劲表现表明，将信息压缩引入除 ARC-AGI 之外的其他数据受限环境（例如药物发现、蛋白质设计），可能有助于我们在这些应用中提取更强的能力。

本文的其余部分介绍了 ARC-AGI 基准测试（第 2 节），详细说明了问题框架（第 3 节），描述了 CompressARC 的架构（第 4 节），展示了实证结果（第 5 节），并以讨论其意义作为结论（第 6 节）。

2 背景：ARC-AGI 基准测试 ARC-AGI-1 是一个人工智能基准测试，旨在测试系统从极少示例中获取新技能的能力 (Chollet, 2019)。 基准测试中的每个谜题都包含一个不同的隐藏规则，系统必须将该规则应用于输入的彩色网格以生成真实的目标彩色网格。 隐藏规则使用了诸如物体性、目标导向性、数字与计数、基本几何和拓扑等主题。 提供了几对输入-输出网格作为示例，以帮助系统找出谜题中的隐藏规则，此外不提供任何其他信息。 图 1 展示了三个 ARC-AGI-1 训练谜题的例子。

图 1：三个 ARC-AGI-1 谜题示例。

- (a) **隐藏规则**：将每个物体向右移动一个像素，物体的底部/右侧边缘除外。
- (b) **隐藏规则**：收缩大物体并将其颜色设置为散落点的颜色。
- (c) **隐藏规则**：延伸绿线以与红线相遇，遇到墙壁时转弯。

虽然基于使用互联网规模数据构建的 LLM 的解决方案在此基准测试中得分达到 87.5% (Chollet, 2024)，且仅在 ARC-AGI 数据上训练的神经网络得分为 40.3% (Wang et al., 2025)，但 CompressARC 将数据最小化的视角推向了极限，选择仅在测试谜题上进行训练。

附录 L 包含关于 ARC-AGI-1 基准测试的更多详细信息。附录 A 中还包含了一份扩展的相关工作调查，包括解决 ARC-AGI 的其他方法。请注意，本文通常将 ARC-AGI-1 简称为 ARC-AGI。

您希望我继续为您翻译接下来的两页（第 3 页和第 4 页）吗?

继续

这里是文档第 3 页和第 4 页的中文翻译。

第 3 页

3 方法

CompressARC 试图解决将数据压缩成尽可能短的程序的问题，以便在保持程序搜索可行性的同时获得谜题的解。 在这种情况下，代码必须是完全自包含的，不接收任何输入，并且必须打印出整个 ARC-AGI 谜题数据集及其填充好的解。通常情况下，运行全算法搜索来找到绝对最短的程序是不可行的，因为我们将不得不搜索海量越来越长的程序，以找到一个输出结果符合我们要求的程序。 CompressARC 通过将自身限制在一个适当且条件良好的程序子空间内，使得程序搜索更加顺畅。 我们的总体策略（将在后续章节详细解释）如下：

• 我们通过一个**模板程序（算法 1）**来定义程序空间，该模板包含许多用于填入硬编码值（“种子”）的空白位置。 这些硬编码的种子将补全程序，使其可以运行，程序运行后将打印出带有解的谜题。

• 我们定义了一个**搜索算法（算法 2）**，它枚举许多可能放入这些空白处的种子，并选择一组好的种子以使算法 1 尽可能短，约束条件是其输出的谜题必须是给定的谜

题，但解的部分是不受约束的。 算法的大部分长度将被种子的长度占据，因此仅需统计种子的总长度即可近似程序长度。

• 算法 2 中只有部分内容对于输出解是必要的；其余部分仅涉及生成谜题的实际压缩表示。 为了解决谜题，我们只保留前一部分，并用**梯度下降过程（算法 3）**来近似搜索的部分步骤。 我们对各种种子长度的可微近似将类似于变分自编码器（VAE）损失函数的各个组成部分 (Kingma & Welling, 2022)。

谜题/解的数据格式：每个谜题采用形状为 `[n_example, width, height, 2]` 的张量形式，包含 $2 \times n\_example$ 个网格中每个像素的颜色指定。 这里， `n_example` 统计谜题中输入/输出网格对的总数，包括那个不知道输出网格的测试网格对。 本节列出的形状仅用于解释目的，实际数据格式将在第 4 节介绍。对数据集进行“代码高尔夫”化的一个天真的初步尝试可能涉及编写一个程序，将每个谜题硬编码在一个巨大的字符串中并将其打印出来。 可以通过巧妙的方法对打印数据中的结构进行去重（例如，引入 for 循环等）来做出改进；我们将在下面详细介绍我们自己的特定策略。

3.1 限制程序空间

我们通过选取一个由**模板程序（算法 1）**组成的程序子空间来开始推导 CompressARC，该模板通过将各种硬编码值替换到指定位置（显示为红色）来完成。 模板程序独立生成每个谜题，并对每个谜题执行以下操作：

1. 它从标准正态分布中随机采样一个形状为 `[n_example, n_colors, width, height, 2]` 的张量 z（第 4 行）。（这里将有一个采样种子，其长度将类似于 VAE 的 KL 损失项。）
2. 使用神经网络 `equivariant_NN` 处理这个张量，输出一个形状为 `[n_example, n_colors, width, height, 2]` 的 logits 张量（第 6 行）。
3. 通过从 logits 张量隐含的概率分布中采样颜色，获得一个形状为 `[n_example, width, height, 2]` 的谜题（第 8 行）。 我们希望这些颜色与给定的谜题相匹配。（这里将再次有一个采样种子，其长度将类似于 VAE 的重构损失项。）

神经网络 `equivariant_NN` 是一个对 ARC-AGI 谜题的有效变换具有“等变性”的神经网络。 这意味着当输入/输出对的顺序或颜色被置换，或者对输入 z 应用翻转或旋转时，神经网络的输出也将以相同的方式变换。

[图 2：CompressARC 近似了一种特定的压缩算法，该算法将 ARC-AGI 谜题数据集转换为能够精确打印它（连同任何解）的最短程序。]

算法 1：生成填好解的完整谜题 P_{filled} 的短程序模板。红色文本（译注：此处用<>表示）需用算法 2 生成的硬编码值替换。

算法 1 代码表：

- **第 1 行：**定义一个 `equivariant_NN` 架构；
- **第 3 行：**设置 seed_z $z =< seed\_z_1 >$（来自算法 2，谜题 1 的硬编码种子）
- **第 4 行：**$z \leftarrow \text{sample}_{seed\_z}(N(0,1))$ （生成输入 z）
- **第 5 行：**设置 $\theta =< \theta_1 >$；（来自算法 2，谜题 1 的硬编码权重）
- **第 6 行：**grid_logits ← equivariant_NN $NN_\theta(z)$；（前向传播）
- **第 7 行：**设置 seed_error $=< seed\_error_1 >$；（来自算法 2，谜题 1 的硬编码种子）
- **第 8 行：**$P_{filled} \leftarrow \text{sample}_{seed\_error}(\text{grid\_logits})$；（生成谜题）
- **第 9 行：**打印 P_{filled}
- **第 11 行：**设置 seed_z $=< seed\_z_2 >$；（...代码对所有谜题重复）

模板程序允许为每个谜题填入两个伪随机采样种子（第 3 行和第 7 行）。 通过操纵第 7 行的最终种子（在第 3 行选择了任意种子之后起作用），可以保证打印出的谜题与数据集中的真实谜题相匹配。有了这个保证，我们可以把模板的代码长度加起来，发现总长度随写下这两个种子所需的比特数/位数而变化。 因此，为了搜索短程序，我们只需要使所有的种子尽可能短。

程序的多个区域可以进行调整以帮助最小化种子长度，我们将在各自的章节中涵盖每一个：第 3、5 和 7 行的种子和权重（下文第 3.2 节），以及第 1 行的架构（第 4 节）。

3.2 种子优化

算法 2 提出了一种优化模板算法 1 中的种子和权重的方法，以减少总种子长度。 它首先尝试操纵模板第 3 行的种子，以模仿从一个不同的学习到的正态分布中采样（第 8 行），然后尝试操纵第二个种子以保证匹配谜题输出（第 11 行）。 然后，它对正态分布参数和神经网络权重执行梯度下降，以最小化总种子长度（第 13-14 行）。

“操纵一个分布的采样种子以模仿从另一个分布采样”的想法往往充满技术细节和微妙之处。 在有利条件下，这可以通过拒绝采样 (Forsythe, 1972) 干净地实现，它使用一个期望长度为两个分布之间最大对数概率比的种子，生成一个来自被模仿分布的样本。 这随后可以通过相对熵编码 (REC) (Harsha et al., 2010; Havasi et al., 2018; Flamich et al., 2021) 改进并扩展到更一般的条件，其中期望种子长度降低到两个分布之间的 KL 散度，代价是只能近似实现所需的采样分布。 我们建议读者参考这些文献以获取详细信息。

我们在算法 2 第 8 行和第 11 行使用 REC 进行种子操纵的主要问题是，运行 REC 所需的时间与被模仿分布和采样分布之间的散度成指数关系 (Flamich et al., 2021)。因此，为了使其更快，我们跳过这些步骤，转而模仿它们的预期下游结果，从而产生了 **CompressARC（算法 3）**。 具体来说，对于... (接下页)

图 2 说明： CompressARC 近似了一种特定的压缩算法，该算法将 ARC-AGI 谜题数据集转换为打印它的最短程序（连同任何解）。 根据奥卡姆剃刀原则，这些打印出的解被假定为实际解的良好预测。 图示流程：

1. **未知 ARC-AGI 解** + **给定 ARC-AGI 谜题** -> 组成 **ARC-AGI 数据集**。
2. **给定 ARC-AGI 谜题** -> 输入到 **Naive Code-Golfer (算法 2)**。
3. **CompressARC (算法 3)** -> 生成 **解**，并逼近 **Naive Code-Golfer (算法 2)**。
4. **Naive Code-Golfer (算法 2)** -> 为 **短程序 (算法 1)** 缩短代码。
5. **短程序 (算法 1)** -> 打印 **ARC-AGI 数据集**。

继续

您希望我继续为您翻译接下来的页面吗?

这里是文档第 5 页和第 6 页的中文翻译。

第 5 页

（接前文）在第 8 行，现在直接从模仿分布中采样 z，算法 2 第 13 行的种子长度被替换为其期望长度，根据 REC 的性质，该长度非常接近模仿分布与采样分布之间的 KL 散度 。对于第 11 行，谜题中未知的网格被直接采样，而第 13 行的种子长度通过采样已知网格的负对数似然（即交叉熵）来密切近似 。

算法 3：CompressARC。它与算法 2 相同，但使用了模拟的种子操纵，并经过截断以返回解出的谜题而非描述。

算法 3 代码表：

- **第 1 行：** 输入：ARC-AGI 数据集；

- **第 2 行**：定义一个 `equivariant_NN` 架构；
- **第 3 行**：对于数据集中的每个谜题 P：
- **第 4 行**：随机初始化 NN_θ 的权重 θ；
- **第 5 行**：观察谜题 P 的维度 `n_example, n_colors, width, height`；
- **第 6 行**：初始化输入分布：形状为 `[n_example, n_colors, width, height, 2]` 的 μ 和对角矩阵 Σ；
- **第 7 行**：执行步骤循环：
- **第 8 行**：$z \leftarrow \text{sample}(N(\mu, \Sigma))$；
- **第 9 行**：grid_logits ⟵ equivariant_NN $NN_\theta(z)$；
- **第 10 行**：$L \leftarrow KL(N(\mu, \Sigma) \mid\mid N(0, I)) + \text{cross-entropy}(\text{grid\_logits}, P)$；
- **第 11 行**：$\mu, \Sigma, \theta \leftarrow Adam(\nabla_\mu L, \nabla_\Sigma L, \nabla_\theta L)$；
- **第 12 行**：结束步骤循环
- **第 13 行**：$P_{filled} \leftarrow \text{sample}(\text{grid\_logits})$；
- **第 14 行**：将 P_{filled} 添加到已解决谜题中
- **第 15 行**：结束谜题循环
- **第 16 行**：返回已解决的谜题

算法 3 (CompressARC) 现在能够通过对模板算法 1 的连续改进来自动压缩 ARC-AGI 数据集，并随后输出解。 唯一剩下的待确定组件是模板算法 1 中使用的神经网络架构，我们将手动进行设计 。

第 6 页

由于架构定义在算法 1 中仅出现一次，而种子为每个谜题重复出现，使用复杂的架构可以帮助我们通过权衡架构描述长度来缩短模板算法 1 的总长度，从而允许更短的种子。 这是我们大力设计神经网络架构的主要动力 。

4 架构

设计神经网络架构的核心思想是创造一种高概率采样 ARC-AGI 谜题的可能性，从而减少种子的长度，并进而缩短模板算法 1 的长度。根据模板结构，这意味着我们需

要神经网络具有良好的归纳偏置（inductive biases），能将噪声转化为看起来合理的ARC-AGI 谜题。 训练谜题在我们的方法中除了增强我们设计层内归纳偏置的努力外，不发挥任何其他作用 。

既然 ARC-AGI 谜题在输入/输出示例顺序、颜色、方向等的任何组合下都具有同等的出现可能性，我们希望我们的网络在默认情况下为它们分配相等的概率。 因此，我们使我们的架构对**示例置换、颜色置换、旋转和翻转**具有等变性 ；从而保证对变换后的输入进行计算会产生同等变换后的输出。 对于谜题可能具有的任何非对称性，我们要求算法 3 操纵随机输入 z 的种子，以偏向输出谜题的某一方面。

图 3：CompressARC 神经网络的核心结构，运行在多张量（multitensor）数据上。

个别操作（彩色）通过通道维度上学习到的投影（灰色）从残差主干读取和写入。 因此，网络对沿其他非通道维度索引的置换具有等变性。 像 `cummax` （累积最大值）这样的一些层会破坏某些几何对称性，赋予架构特定的几何能力（详见附录 H)。图中未显示归一化、softmax、平移（shift）和方向层。

该架构由一个功能类似于嵌入矩阵的解码层（详见附录 D.1)、一个带有残差主干的核心以及通道维度上的线性读出（见附录 D.8）组成。 在核心部分，通道维度上的线性投影从残差中读取数据到专门设计的操作中，这些操作通过另一个投影将输出写回残差。 归一化操作分散在各层中，然后整个核心层块重复 **4 次**。 这非常像Transformer 架构 ，不同之处在于专门设计的操作不是针对序列的注意力和非线性操作，而是针对谜题形状数据的以下操作：

- 沿轴对一个张量求和并/或将结果广播回去（见附录 D.2)。
- 沿张量的一个或多个轴进行 softmax（见附录 D.3)。
- 沿张量的一个几何维度取累积最大值（见附录 D.4)。
- 沿张量的一个几何维度平移一个像素（见附录 D.4)。

您希望我继续为您翻译接下来的第 7 页和第 8 页吗?

继续

这里是文档第 7 页和第 8 页的中文翻译。

第 7 页

- 逐元素非线性操作（见附录 D.6）。
- 沿通道维度的归一化（见附录 D.7）。
- 以及附录 D.5 中描述的另一种操作。这些操作本身没有参数，其行为由残差层的读/写权重控制 。 所有这些读/写投影都在通道维度上运行 。 我们在主干的某些部分使用 16 个通道，在其他部分使用 8 个通道，以降低计算成本 。 由于这些投影占据了模型权重的大部分，整个模型仅有 **7.6 万（76K）个参数** 。

神经网络用于计算的实际数据格式不是形状为 `[n_example, n_colors, width, height, channel]` 的单个张量，而是一组**多张量（multitensor）**，每个张量具有这些维度的不同子集（例如 `[n_colors, width, channel]` 张量）。 网络的输入 z、输出的 logits 以及所有的内部激活都采用多张量的形式 。 通常，对于这些维度的每一个子集都有一个张量用于存储该形状的信息，这有助于构建有用的归纳偏置 。 例如，网格列到颜色的分配可以作为独热表存储在形状为 `[color, width, channel]` 的张量中 。 关于多张量的更多细节见附录 C 。

5 结果

我们为 CompressARC 在每个谜题上提供了 2000 个推理时训练步骤，每个谜题耗时约 20 分钟 。 在此推理计算预算内，CompressARC 正确解决了 **20% 的评估集谜题和 34.75% 的训练集谜题** 。 图 4 展示了性能随推理时间增加而提升的情况 。 附录中的表 4 和表 5 记录了数值求解准确率和计时 。

表 1：各种 ARC-AGI-1 方法的求解准确率对比（按使用的训练数据量排序）。 每种方法对每个谜题进行两次解的猜测，只有当网格形状和像素颜色全部正确时，该猜测才被视为正确 。

方法	训练于:	神经	准确率	数据集划分
随机猜测	无	X	0%	全部
规则暴力搜索 (Kamradt, 2024)	无	X	40%	私有评估集
U-Net 基准模型	目标谜题	✓	0.75%	公开评估集
CompressARC (本研究)	**目标谜题**	✓	**20%**	**公开评估集**
HRM 消融实验	测试谜题	✓	31%	公开评估集
HRM (Wang et al., 2025)	训练+测试谜题	✓	40.3%	公开评估集
OpenAI o3 high (Chollet, 2024)	互联网规模数据	✓	87.5%	半私有评估集

5.1 我们可以解决和不能解决哪些谜题? CompressARC 尝试利用其能力尽可能多地解决谜题，直到遇到其能力的瓶颈 。 例如，训练集中的谜题 `28e73c20` 要求将模式从边缘延伸向中间（见图 12a）。 基于现有的网络层，CompressARC 通常能够进行短距离的模式延伸，但无法进行长距离延伸 。 因此，它尽其所能，正确延伸了一小段距离，然后开始猜测中心附近的情况（图 12b）。

第 8 页

图 4：CompressARC 的谜题解决准确率随着推理时学习的进行而上升。 展示了用于准确率测量的不同允许猜测次数 (pass@n) 。 官方基准测试以 2 次允许猜测为准报告，这也是我们在评估集上报告 20% 的原因 。 (a) 400 个训练集谜题；(b) 400 个评估集谜题 。

5.2 案例研究：为方块着色 在所示谜题（图 5）中，必须根据方块所在的网格位置为其着色 。 我们称这个谜题为“为方块着色”(Color the Boxes) 。

图 5：为方块着色，谜题 272f95fa。

5.2.1 解决方案分析

人类解决方案：我们首先意识到输入被划分为方块，输出中方块仍然存在，但现在被着色了 。 然后我们尝试找出哪些颜色对应哪些方块 。 首先，我们注意到角落总是黑色。然后，我们注意到中间总是洋红色 。 之后，我们注意到侧边方块的颜色取决于它们的方向：上红、下蓝、右绿、左黄 。 此时，我们将输入复制到答案网格，然后将中间方块涂成洋红色，接着根据方向为其余方块着色 。

CompressARC 解决方案：表 2 展示了 CompressARC 随时间推移的学习行为 。 在学习完成后，我们可以拆解其学到的 z 分布，发现它编码了一个“颜色-方向”对应表和行/列分割位置（图 7）。 在训练期间，重构误差下降得极快。它保持在较低的平均水平，但偶尔会大幅波动，导致图 6a 所示的 z 的 KL 散度在这些时刻向上跳变 。

我们观察 z 中存储的表示，以了解 CompressARC 如何学习解决此谜题 。 由于 z 是一个多张量，它包含的每个张量都对 z 的总 KL 产生累加贡献。通过观察每个张量的贡献（见图 6b），我们可以确定 z 中的哪些张量编码了用于表示谜题的信息 。 在训练期间，除了四个张量外，所有张量的信息内容都降至零 。 在该实验的一些重复运行中，我们看到这四个必要张量之一也会降至零，通常之后 CompressARC 就无法恢复正确答案 。 这里我们展示的是一次幸运的运行，其中 `[color, direction, channel]` 张量几乎掉落但在 200 步时被重新拾起，这恰好是模型样本开始显示正确颜色和位置的时候 。

您希望我继续为您翻译接下来的第 9 页和第 10 页吗?

这里是文档第 9 页和第 10 页的中文翻译：

第 9 页

表 2：CompressARC 学习“为方块着色”谜题解法的过程。

学习步数	CompressARC 在做什么？	采样的解决方案预测
50	CompressARC 的网络输出一个答案网格（样本），在输入网格有浅蓝色行/列的地方，它也输出浅蓝色。它注意到谜题中所有其他输入-输出对都展现了这种对应关系 。它不知道其他输出像素如何分配颜色；网络输出的指数移动平均（样本平均值）显示网络对非浅蓝色像素分配了大部分相同的平均颜色 。	样本 (sample) / 样本平均值 (sample average)
150	网络输出一个网格，其中相邻像素具有相似的颜色 。它可能注意到这在所有输出中都很常见，并猜测这同样适用于答案 。	样本 / 样本平均值
200	网络输出现在显示出被浅蓝色边界隔开的大块颜色 。它注意到了在其他输出中常用边界来划分颜色块的做法，并在此应用了同样的想法 。它还注意到了其他给定输出中的黑色角落色块，网络对此进行了模仿 。	样本 / 样本平均值
350	网络输出现在显示了根据与中心的方向分配给方块的正确颜色 。它意识到在其他给定输出中使用了单一的“颜色-方向”映射来选择色块颜色，因此它模仿了这种映射 。它在行内着色方面仍不是做得最好，对中心色块也感到困惑，可能是因为中间不对应任何方向 。尽管如此，平均网络输出在中间显示出一抹正确的洋红色，这意味着网络正在领悟 。	样本 / 样本平均值
1500	网络已经达到了最精炼的状态 。有时它在输出的样本中仍会犯错，但这并不常见且会被过滤掉 。	样本 / 样本平均值

第 10 页

图 6：训练期间损失成分的分解告诉我们 CompressARC 偏好在哪里以及如何存储解决谜题的相关信息。

- **(a)** 训练期间 KL 项和重构项在损失中的相对比例（在取加权和之前）。KL 项主导了损失，而重构通常几乎是完美的。
- **(b)** 将训练期间的 KL 损失分解为多张量 z 中每个独立形状张量的贡献。四个张量占据主导地位，表明它们包含信息，而其他 14 个张量降至零，表明它们缺乏信息内容。

图 7：通过分解训练期间的损失成分，揭示 CompressARC 存储谜题信息的偏好。

- **(a) [example, height, channel] 和 [example, width, channel] 张量**。对于每个示例和行/列，都有一个维度为 $n\_channels$ 的向量 。对这组向量进行主成分分析（PCA），顶层主成分（在两个张量中都比其他成分强 1000 倍以上）的可视化结果告诉我们哪些“示例/行”和“示例/列”组合是由存储的信息唯一识别的。对于每个示例，顶层矩阵中最亮的两个像素给出了谜题网格中浅蓝色行的位置，底层矩阵中最暗的两个像素指示了列。

- **(b) [direction, color, channel] 张量**。采用类似的 PCA 分解：图表显示了该张量的顶层主成分。四个最亮的像素识别出：蓝色对应上，绿色对应左，红色对应下，黄色对应右。该张量告诉每个方向在对向边缘的方块中使用哪种颜色。顶层主成分比次优主成分强 829 倍。

6 讨论 现代深度学习对高质量数据的普遍依赖，使得该领域在应用于数据较少的智能行为问题时陷入了僵局。 这种情况在数据受限的 ARC-AGI 基准测试中尤为明显，目前该基准测试由在专门增强、扩展和策划的数据集上训练的 LLM 主导 。

在这种背景下，我们构建了 CompressARC，它不仅不使用任何训练数据，甚至放弃了整个预训练过程。 直觉上人们会预料这会失败且解不出任何谜题，但通过在推理阶段对目标谜题应用 MDL，CompressARC 解决了 ARC-AGI-1 中令人惊讶的一大部分题目。

您希望我继续为您翻译接下来的第 11 页和第 12 页吗?

继续

这里是文档第 11 页和第 12 页的中文翻译：

第 11 页

CompressARC 的理论基础源于最小化目标谜题的程序化描述长度 。 虽然由于可能的程序搜索空间巨大而导致其他 MDL 搜索策略稀缺，但 CompressARC 通过梯度下降探索了一个简化的、基于神经网络的搜索空间 。 尽管 CompressARC 的架构经过了高度工程化，但它仅凭两个演示输入/输出对就能进行泛化的惊人能力，使其在 ARC-AGI 的泛化领域进入了一个全新的境界 。

提高效率仍然是 CompressARC 未来研究的一个重要方向 。 CompressARC 使用了许多自定义操作，添加 JIT 编译内核或融合 CUDA 内核将提高训练迭代速度 。 鉴于我们的架构运行时间随谜题中的像素数量而变化，这些改进自然会对更大的网格产生更显著的影响 。

我们对“智能必须源于大规模预训练和数据”的假设提出了挑战，并展示了巧妙利用 MDL 和压缩原理可以产生令人惊讶的能力 。 我们以 CompressARC 为概念验证，证明了现代深度学习框架可以与 MDL 融合，从而创造出一种通往通用人工智能（AGI）的可能的替代性、补充性路径 。

第 12 页：参考文献

参考文献

- Akyürek 等人 (2024). 测试时训练（Test-time training）在少样本学习中的惊人效果。
- ARC Prize 团队 (2025). HRM 在 ARC-AGI 上表现的隐藏驱动力。
- Barbadillo (2024). ARC24 解决方案摘要。
- Barbadillo (2025). 探索 ARC25 挑战赛中搜索与学习的结合。
- Berman (2024). 我如何在 ARC-AGI 上获得创纪录的 53.6% 分数。
- Bonnet 和 Macfarlane (2024). 搜索潜程序空间。
- Bretagnolle 和 Huber (1978). 密度估计：极小极大风险。
- Chauhan 等人 (2024). 深度学习中超网络（Hypernetworks）的简要回顾。
- Chollet (2019). 论智能的衡量。
- Chollet (2024). OpenAI o3 在 ARC-AGI 公开榜单上的突破性高分。

- Cohen 和 Welling (2016). 群等变卷积网络（Group equivariant convolutional networks）。
- Cole 和 Osman (2025). 不要因噎废食：ARC 深度学习的方法与原因。
- Fan 等人 (2021). 热带卷积（Tropical convolution）替代传统卷积神经网络的一种实践。
- Flamich 等人 (2021). 通过相对熵编码（REC）对潜表示进行编码来压缩图像。
- Forsythe (1972). 冯·诺依曼的正态分布及其他分布随机采样的比较方法。
- Graves 等人 (2014). 神经图灵机（Neural Turing Machines）。

是否需要我继续翻译接下来的第 13 页和第 14 页?

继续

这里是文档第 13 页和第 14 页的中文翻译：

第 13 页

- Greenblatt (2024). 使用 GPT-4o 在 ARC-AGI 上达到 50% (SOTA)。
- Harsha 等人 (2010). 相关性的通信复杂度。
- Havasi 等人 (2018). 最小随机代码学习：从压缩模型参数中获取比特位。
- He 等人 (2015). 用于图像识别的深度残差学习（ResNet）。
- Hendrycks 和 Gimpel (2023). 高斯误差线性单元 (GELUs)。
- Higgins 等人 (2017). beta-VAE：使用受限变分框架学习基本视觉概念。
- Hodel (2024). 用于抽象和推理语料库（ARC）的领域特定语言（DSL）。

- Hu 等人 (2021). LoRA：大语言模型的低秩自适应。
- Hutter (2005). 通用算法智能体 AIXI。
- Hutter (2006). 赫特奖（Hutter Prize）：人类知识的无损压缩。
- Kamradt (2024). ARC Prize 2024 解决方案：第 4 名，得分 40。
- Kingma 和 Welling (2022). 变分自编码器（Auto-encoding variational bayes）。
- Knoop (2024). ARC Prize 2024 获奖者与技术报告发布。
- Kolmogorov (1998). 论随机数表。
- Langdon (1984). 算术编码简介。
- LeGris 等人 (2024). H-ARC：对 ARC 基准测试中人类表现的稳健估计。
- Li 等人 (2024a). 结合归纳和传导进行抽象推理。

第 14 页

- Li 等人 (2024b). 结合归纳和传导进行抽象推理（arXiv 版本）。
- Odouard (2024). ARC 解决方案文档。
- Rissanen (1978). 通过最短数据描述建模（MDL）。

- Ronneberger 等人 (2015). U-Net：用于生物医学图像分割的卷积网络。
- Shannon (1956). 有噪信道的零误差容量。
- Shannon (1948). 通信的数学理论。
- Solomonoff (1964). 归纳推理的形式理论（第一部分）。
- Sun 等人 (2020). 通过自监督进行测试时训练（Test-time training），以实现在分布偏移下的泛化。
- Sønderby 等人 (2016). 阶梯变分自编码器。
- Tsybakov (2008). 非参数估计导论。
- Vahdat 和 Kautz (2021). NVAE：深度分层变分自编码器。
- van den Oord 等人 (2018). 神经离散表示学习（VQ-VAE）。
- Vaswani 等人 (2023). 注意力机制就是你需要的全部（Attention is all you need）。
- Wang 等人 (2025). 分层推理模型（Hierarchical reasoning model）。
- Xiong 等人 (2020). 论 Transformer 架构中的层归纳化。
- Zaheer 等人 (2018). 深度集合（Deep Sets）。

需要我继续翻译接下来的附录部分（第 15 页和第 16 页）吗?

继续

这里是文档第 15 页和第 16 页的中文翻译：

第 15 页

附录 A 相关工作

A.1 压缩与智能的等价性 本研究的最初灵感来自赫特奖（Hutter Prize）。该奖项旨在奖励那些能将维基百科文本文件压缩到极致的人，以此激励研究人员构建智能系统 。其前提是认为压缩信息的能力等同于智能 。这种等价关系有着悠久的历史 。例如，在讨论预测问题的智能解决方案时，理想的预测器会实现所罗门诺夫归纳法（Solomonoff Induction），这是一种理论上最佳但不可计算的算法，适用于所有预测任务 。这种预测算法等同于最佳压缩算法，其压缩代码长度即为数据的柯尔莫哥洛夫复杂度（Kolmogorov Complexity）。该算法还可用于解码数据的最小长度描述，从而将智能的表述与最小描述长度（MDL）联系起来 。在我们的工作中，我们尝试用神经网络来近似这种最佳压缩算法 。

A.2 信息论与编码理论 由于我们构建的是信息压缩系统，因此利用了信息论和编码理论的诸多成果 。激励我们模型架构的主要结果是相对熵编码（REC）的存在性 。REC 的存在意味着只要 KL 散度是有界的，压缩算法的构建总是可能的，具体算法实现的细节可以被抽象掉 。因此，关于编码理论以及如何将信息从高斯分布翻译成二进制（及反向翻译）的问题可以被忽略，因为我们可以直接从高斯分布计算二进制代码长度 。换句话说，我们只需要利用高斯分布进行足够的信息论推导即可完成任务，无需涉及具体的编码理论 。虽然算术编码在离散分布下已足够，但神经网络运行在连续空间，因此我们需要 REC 。我们的架构通过加性高斯白噪声（AWGN）信道发送 z 信息，因此 AWGN 信道容量公式（高斯输入高斯噪声）在我们的解码层设计中起到了关键作用 。

A.3 变分自编码器 变分自编码器（VAE）的解码器端充当我们的解压缩算法 。虽然我们本可以使用具有更通用能力的神经图灵机，但其并不利于基于梯度下降的优化，因此我们坚持使用 VAE 。VAE 的许多发展与我们相关 。我们曾尝试使用多个

解码层构建分层 VAE 解码器 ，但发现第一层会吸收所有的 KL 贡献，使后续层失效 。因此我们最终只在开始处使用一个解码层 。beta-VAE 引入了重构损失的加权，使其强于 KL 损失，这在我们的案例中效果良好 。NVAE 应用了非恒定的损失组件权重 。CompressARC 中使用了一种初级的计划损失重组 。

A.4 其他 ARC-AGI 方法 得分最高的方法通常依赖于将谜题网格转换为文本，然后输入到预训练的 LLM 中以寻找答案 。主要技术包括直接输出解网格 或输出可运行的程序来操作网格 。

第 16 页

这些方法通常采用以下技巧来提高性能：

- **微调**：在训练谜题数据上进行微调 。

- **数据增强**：增加微调谜题的有效数量 。

- **合成数据**：在合成数据上微调 。

- **推理时训练**：在测试时针对每个测试谜题微调特定模型 。

- **测试时训练 (TTT)**：利用 TTT 技术 。

- **集成学习**：采样多个模型输出或随机增强版本进行集成 。

- **LLM 推理**：利用 LLM 的推理能力 。

这些方法得分高达 87.5%，但每个谜题的推理计算成本超过 200 美元 。它们都使用了在整个互联网上预训练的语言模型，而 CompressARC 仅使用测试谜题作为训练

数据 。它们的共同点在于强调在单个测试谜题上训练个体模型以及使用集成方法 。此外还有：

- **代码搜索**：在手工设计的领域特定语言（DSL）中进行大规模程序搜索 。
- **潜程序空间搜索**：引入基于 VAE 的方法搜索程序潜空间，这是与我们最相似的工作 。

A.5 深度学习架构 我们从头设计了架构，但也借鉴了许多原则 。结构上类似于 Transformer，包含残差流和线性头 。

- **读写机制**：通过线性上/下投影进行前置和后置归一化（Pre/Post-norm），允许各层读写残差流 。
- **激活函数**：使用类似于 Transformer 的 SiLU 非线性层 。
- **等变性**：受置换不变神经网络启发，采用等变架构 。等变变换取自 ARC-AGI 谜题的常见增强方式 。

附录 B 通过 KL 和交叉熵进行种子长度估算 在 3.1 节中，我们将算法 2 第 12 行的 $len(seed\_z)$ 估算为 $KL(N(\mu,\Sigma) \mid\mid N(0,I))$，并将 $len(seed\_error)$ 估算为交叉熵 。Flamich 等人 (2021) 证明，如果存在共享随机源 P，种子通信是通信分布 Q 样本最有效的方式，预期每个样本消耗约 $KL(P \mid\mid Q)$ 比特 。

您希望我继续为您翻译接下来的第 17 页和第 18 页吗?

继续

这里是文档第 17 页和第 18 页的中文翻译：

第 17 页

读者可以参考 Flamich 等人 (2021) 关于操作流程、运行时间和内存分析以及近似强度的详细内容。 以下是我们自己对原因的推理尝试。 简而言之，我们在算法 2 中的原始程序是操纵采样 $z \sim N(0, I)$ 的种子，以模拟 $z \sim N(\mu, \Sigma)$ 的采样，我们希望证明可以使用接近 $KL(N(\mu, \Sigma) \mid\mid N(0, I))$ 的预期种子比特数来密切近似这种采样。

为了便于说明，假设算法 2 实现了一种类似于拒绝采样 (Forsythe, 1972) 的机制，逐个迭代种子并以概率 $min(1, cw(z))$ 接受样本，其中 $c \ll 1$ 且 $w(z)$ 是概率比：

$$w(z) = \frac{N(z; \mu, \Sigma)}{N(z; 0, I)}$$

当我们选取足够小的 c 时，采样分布将如预期般变得任意接近 $N(\mu, \Sigma)$。 在此 c 下，我们希望证明预期拒绝次数产生的种子长度接近 KL 散度。 我们首先给出每一步接受概率 P_{accept} 的下界：

$$P_{accept} = \int N(z; 0, I) \min(1, cw(z)) dz = \int \min(N(z; 0, I), cN(z; \mu, \Sigma)) dz$$

我们将遵循 Tsybakov (2008) 对 Bretagnolle-Huber 不等式 (Bretagnolle & Huber, 1978) 的修改推导来得出 KL 的界限：

$$(1 + c)P_{accept} \geq (1 + c - P_{accept})P_{accept}$$

$$= \left(\int \max(N(z; 0, I), cN(z; \mu, \Sigma)) dz\right) \left(\int \min(N(z; 0, I), cN(z; \mu, \Sigma)) dz\right)$$

应用带有函数空间内积的 Cauchy-Schwarz 不等式：

$$\geq \left(\int \max(N(z; 0, I), cN(z; \mu, \Sigma)) \min(N(z; 0, I), cN(z; \mu, \Sigma)) dz\right)^2$$

$$= (\int \quad cN(z;0,I)N(z;\mu,\Sigma) \quad dz)^2$$

$$= c\exp(2\ln\int \quad N(z;0,I)N(z;\mu,\Sigma) \quad dz) = c\exp(2\ln \mathrm{E}_{z\sim N(z;\mu,\Sigma)}[\quad \frac{N(z;0}{N(z;\mu}$$

随后应用 Jensen 不等式：

$$\geq c\exp(\mathrm{E}_{z\sim N(z;\mu,\Sigma)}[\ln\frac{N(z;0,I)}{N(z;\mu,\Sigma)}]) = c\exp(-KL(N(\mu,\Sigma) \mid \mid N(0,I)))$$

这导出了至少为以下的接受概率：

$$P_{accept} \geq \frac{c}{1+c}\exp(-KL(N(\mu,\Sigma) \mid \mid N(0,I)))$$

第 18 页

因此，根据拒绝采样程序，提议的样本数量（即预期种子）最多是该接受概率的倒数：

$$seed\_z \leq \frac{1+c}{c}\exp(KL(N(\mu,\Sigma) \mid \mid N(0,I)))$$

所以预期的种子长度最多约为对数值：

$$len(seed\_z) \leq KL(N(\mu,\Sigma) \mid \mid N(0,I)) + \log(1+c) - \log c$$

这与我们陈述的种子长度 KL 近似值相匹配。

对于 `seed_error` 项，算法 2 操纵种子以从某些 logits 隐含的分布中采样谜题 P 。 这实际上等同于采样 $grid\_logits \sim \text{Categorical\_distribution}(logits)$ 并操纵种子试图让 $grid\_logits$ 接近 $\text{Delta\_distribution}(P)$。 然后，可以再次使用相同的基于 KL 的种子长度界限。 预期的 `seed_error` 长度最多为：$KL(\text{Delta\_distribution}(P) \mid \mid \text{Categorical\_distribution}(logits)) + \log(1+c) - \log c$ 简化为：

$$= \text{cross\_entropy}(logits, P) + \log(1+c) - \log c$$

附录 C 多张量 (MULTITENSORS) 穿过我们各层级的实际数据（z、隐藏激活值和谜题）采用一种我们称为“多张量”的格式。 如**图 8** 所示，它只是一个包含各种形状张量的桶。我们使用的所有等变性都可以用它们如何改变多张量来描述。

大多数常见的机器学习架构在单一类型的恒定秩张量上操作。 LLM 在形状为 `[n_batch, n_tokens, n_channels]` 的 3 秩张量上运行，而 CNN 在形状为 `[n_batch, n_channels, height, width]` 的 4 秩张量上运行。 我们的多张量是一组独特类型的变秩张量，其维度是形状为 `[n_example, n_colors, n_directions, height, width, n_channels]` 的 6 秩张量的子集。

是否需要我继续翻译接下来的第 19 页和第 20 页?

继续

这里是文档第 19 页和第 20 页的中文翻译：

第 19 页

我们始终保留通道维度，因此每个多张量中最多有 32 个张量 。我们还维护了几条规则（见附录 E.1）来确定张量形状是否“合法”，这使多张量中的张量数量减少到 18 个 。

表 3：多张量维度的尺寸约定

维度	描述
示例 (Example)	ARC-AGI 谜题中的示例数量，包括留出答案的那一个。
颜色 (Color)	谜题中独特颜色的数量，不包括黑色（见附录 F.2）。
方向 (Direction)	8 个方向，在预处理谜题时确定（见附录 F.1）。
高度 (Height)	在预处理谜题时确定（见附录 F.1）。
宽度 (Width)	在预处理谜题时确定（见附录 F.1）。
通道 (Channel)	在残差连接中，如果包含方向维度则尺寸为 8，否则为 16。

为了说明多张量如何存储数据，可以通过使用 `[example, color, height, width, channel]` 张量来表示一个 ARC-AGI 谜题。利用通道维度选择输入或输出网格，高度/宽度维度表示像素位置，颜色维度使用独热向量（one-hot）来指定该像素的颜色。 `[example, height, channel]` 和 `[example, width, channel]` 张量可类似地用于存储代表每个示例网格形状的掩码。

当我们在多张量上应用操作时，默认假设所有非通道维度都被视为批处理（batch）维度。除非另有说明，否则操作会跨维度索引进行复制，这确保了在我们使用专门用于破坏对称性的层之前，所有的对称性都保持完整。

附录 D 架构中的各层

D.1 解码层 (DECODING LAYER) 该层的作用是采样一个多张量 z 并限制其信息内容，然后将其传递到下一层。该层输出学习到的 z 分布与 $N(0,I)$ 之间的 KL 散度

。惩罚 KL 散度可以防止 CompressARC 以未压缩的方式记忆谜题，迫使网络更简洁地表示谜题 。

具体来说，每当网络使用 z 来破坏某种对称性时，该层就会迫使网络在 KL 项上消耗更多比特；破坏的对称群越大，消耗的比特就越多 。

第 20 页

D.2 多张量通信层 (MULTITENSOR COMMUNICATION LAYER) 该层允许一个多张量中的不同张量相互交互 。首先，来自残差流的输入通过每张量投影转换为固定大小（向下通信为 8，向上通信为 16） 。然后，消息被发送到具有“至少相同维度”(向上通信）或“至多相同维度”(向下通信）的每个其他张量 。

如图 9 所示，这种消息是通过沿维度取平均值以移除维度（降秩），或通过取消挤压（unsqueeze）和广播（broadcast）维度以增加维度来创建的 。每个张量接收到的所有消息会被求和并进行归一化 。结果经过上投影（up-projected）后加回到残差流中 。

图 9：多张量通信层。高秩张量显示在顶部，低秩显示在底部。张量通过均值约简（mean reduction）和取消挤压维度在不同秩之间转换 。

D.3 SOFTMAX 层 该层允许网络处理内部的独热表示，通过提供去噪和锐化带噪声独热向量的工具来实现 。对于输入多张量中的每个张量，该层列出所有可能的维度子集进行 softmax 操作，并将所有 softmax 后的结果在通道维度上拼接 。输出维度因多张量中不同张量的秩而异 。该层采用了前置归一化（pre-norm），并通过每张量投影与残差流进行映射 。

您希望我继续为您翻译接下来的第 21 页和第 22 页吗?

继续

这里是文档第 21 页和第 22 页的中文翻译：

第 21 页

并在通道维度上拼接所有 Softmax 后的结果。根据张量的秩，多张量中不同张量的输出维度会有所不同。该层应用了前置归一化（pre-norm），并通过每张量投影与残差流进行映射。该层的输入通道维度为 2。

D.4 定向累积最大值/平移层 (DIRECTIONAL CUMMAX/SHIFT LAYER) 定向累积最大值（Cummax）和平移（Shift）层允许网络以等变的方式执行非等变的 Cummax 和 Shift 操作，即每次针对一个方向应用操作，且只有在方向通过多张量通信层重新聚合后才允许输出受到结果的影响。这些层正是我们在定义多张量时包含“方向”维度的唯一原因：用于存储定向层的结果并对其进行单独操作。当然，这意味着当我们应用空间等变变换时，我们也必须相应地置换方向维度的索引，这有时会变得很复杂。

定向 Cummax 层获取方向维度的 8 个索引，将每个切片视为对应一个方向（4 个基数方向，4 个对角方向），在每个切片的相应方向上执行累积最大值，对一半的通道在相反方向上执行相同操作，并将切片重新堆叠回方向维度。图 10 给出了示意图。在应用累积最大值之前，切片被重新缩放至最小值 -1 和最大值 1 之间。

定向平移层执行相同的操作，但是将网格平移一个像素，而不是应用累积最大值，且不进行重新缩放。

一些细节：

- 通过每张量投影与残差流进行映射，带有前置归一化。
- 输入通道维度为 4。
- 这些层仅应用于输入多张量中的 `[example, color, direction, height, width, channel]` 和 `[example, direction, height, width, channel]` 张量。

图 10：定向 Cummax 层。该操作有助于 CompressARC 在谜题网格中跨越长距离传输信息。

D.5 定向通信层 (DIRECTIONAL COMMUNICATION LAYER) 默认情况下，网络对 8 个方向的置换是等变的，但我们只想要旋转和翻转下的对称性。因此，该层提供了

一种根据两个方向之间的角度差，在方向维度的两个切片之间发送信息的方法。

第 22 页

该层为 64 种可能的角度组合分别定义了一个线性映射，但线性映射的权重被最小限度地绑定，以确保定向通信层对反射和旋转具有等变性。由于在应用等变变换时，方向维度的索引也会置换，这变得非常复杂。张量中的每个方向切片累积其 8 条消息，并将结果相加。该层通过每张量投影与残差流映射，带有前置归一化。输入通道维度为 2。

D.6 非线性层 (NONLINEAR LAYER) 我们使用通道维度为 16 的 SiLU 非线性激活函数，并由带有前置归一化的每张量投影包围。

D.7 归一化层 (NORMALIZATION LAYER) 我们对多张量中的所有张量进行归一化，使用跨除通道维度外所有维度计算的均值和方差。其他层内部使用的归一化通常也以这种方式运行。

D.8 线性头 (LINEAR HEADS) 我们必须获取最终的多张量，并将其转换为 ARC-AGI 谜题的格式。更具体地说，我们必须将多张量转换为 ARC-AGI 谜题上的分布，以便计算观察到的谜题网格的对数似然。

图 11：线性头层从残差流的最终多张量中读取数据，解释为颜色 Logits 和形状掩码。

每个示例的输入和输出像素颜色由 `[example, color, height, width, channel]` 张量定义 Logits，通道维度线性映射为 2，代表输入和输出网格。对数似然由交叉熵给出，并在所有网格上进行求和规约。

我们还根据角度乘以系数：0 度和 180 度为 1，45 度和 135 度为 0.2，90 度为 0.4。线性映射初始化为输入和输出网格相同，但在学习过程中不固定。

您希望我继续为您翻译接下来的第 23 页和第 24 页吗?

继续

这里是文档第 23 页和第 24 页的中文翻译：

第 23 页

对于非固定形状的网格，使用 `[example, height, channel]` 和 `[example, width, channel]` 张量来创建每个颜色网格可能存在的连续矩形切片的分布，如图 11 所示 。同样，通道维度被映射为大小 2，分别对应输入和输出网格 。对于每个网格，我们都有一个大小为 `[width]` 的向量和一个大小为 `[height]` 的向量 。每一个切片的对数似然被取为该切片内数值的总和减去切片外数值的总和 。然后将所有可能切片的对数似然归一化，使总概率为 1，每个切片的颜色由前一段描述的颜色 Logits 定义 。定义好谜题分布后，我们现在可以评估观测到的目标谜题的对数似然，将其作为重构误差 。

E 其他架构细节

E.1 合法多张量的规则

1. 必须包含至少一个非示例（non-example）维度 。示例除了与颜色、方向、行和列相关之外，并没有其他特殊性 。

2. 如果包含了宽度或高度维度，则也应包含示例维度 。位置是网格固有的属性，而网格是由示例维度索引的 。如果没有网格，谈论位置就没有多大意义 。

E.2 针对反射/旋转对称的权重绑定 当对多张量中的每个张量应用不同的线性层时，我们会为具有宽度但没有高度维度的张量设置一个线性层，为具有高度但没有宽度维度的张量设置另一个线性层 。只要出现这种情况，我们就会将它们的权重绑定在一起，以保持整个网络对对角线反射和 90 度旋转的等变性（这些变换会交换宽度和高度维度）。Softmax 层没有完全对称化，因为输出的不同索引对应于不同的 Softmax 维度组合 。正确绑定这些权重会非常复杂且耗时，相对于预期的性能提升并不划算，因此我们没有这样做 。

E.3 训练/初始化 我们使用 Adam 优化器进行 2000 次迭代训练，学习率为 0.01，β_1 为 0.5，β_2 为 0.9 。权重基本上都使用 Xavier 正态分布初始化（Xavier normal initialization） 。

F 预处理

F.1 输出形状判定 原始数据由各种形状的网格组成，而神经网络运行在固定形状的网格上 。我们做的大部分预处理都是为了解决这种形状不一致的问题 。在进行任何训

练之前，我们会判定给定的 ARC-AGI 谜题是否遵循以下三种可能的形状一致性规则：

1. 给定谜题中的输出形状始终与相应的输入形状相同 。

2. 给定谜题中的所有输入形状都相同 。

第 24 页

3. 给定谜题中的所有输出形状都相同 。

基于规则 1 和 3，我们尝试预测留出示例的输出形状，优先考虑规则 1 。如果任一规则成立，我们会通过覆盖线性头层产生的掩码，强制后处理步骤仅考虑预测的形状 。如果规则均不成立，我们会临时预测给定谜题中所有网格的最大宽度和高度，并允许掩码预测比该尺寸更小的形状 。给定或预测的最大宽度和高度将作为多张量宽度和高度维度的尺寸 。预测的形状也作为掩码用于多张量通信、定向通信以及定向累积最大值/平移层 。由于时间限制以及我们认为这不会带来太多性能提升，我们没有在其他层应用掩码 。

F.2 颜色数量 我们注意到，在几乎所有的 ARC-AGI 谜题中，谜题中未出现的颜色也不会出现在真实答案中 。因此，任何未在谜题中出现的颜色都不会在多张量的颜色维度中分配索引 。此外，黑色被视为特殊颜色，不包含在多张量中，因为它通常在许多谜题中代表背景 。进行颜色分类时，会在应用线性头后在颜色维度追加一个全零张量，以代表黑色的 Logits 。

G 后处理

由于生成的答案网格受 z 的随机性影响是随机的，我们在整个训练过程中保存答案网格，粗略地说，我们选择出现频率最高的一个作为去噪后的最终预测 。这在处理某些谜题中的变长形状网格时变得更加复杂 。通常，当我们通过提取颜色维度 Logits 并取最大值（argmax）从网络采样答案时，网格会带有噪声，且常在几个随机像素上出现错误颜色 。我们开发了几种去除此类噪声的方法：

1. 寻找采样次数最多的答案 。

2. 在进行 Softmax 生成概率之前，构建输出颜色 Logits 的指数移动平均 。同时构建掩码的指数移动平均 。

3. 在进行 Softmax 之后，构建输出颜色概率的指数移动平均 。同时构建掩码的指数移动平均 。

应用这些技术时，我们总是取掩码下概率最高的切片，然后取该切片中概率最高的颜色 。我们探索了几种选择哪种方法的规则，最终采用了方法 1 和 2 的组合：

- 在每次迭代中，统计采样答案以及指数移动平均答案（衰减率 $decay = 0.97$ ） 。

- 如果在训练的前 150 次迭代内，则将该答案权重降低 e^{-10} 倍（实际上是不计入该答案） 。

- 如果答案来自指数移动平均而非单次采样，则将权重降低 e^{-4} 倍 。

您希望我继续为您翻译接下来的第 25 页和第 26 页吗?

继续

这里是文档第 25 页和第 26 页的中文翻译:

第 25 页

- **** 根据 $e^{-10\times\text{uncertainty}}$ 降低答案的权重，其中 uncertainty（不确定性）是每个像素被分配给最高概率颜色的负对数概率的平均值（跨所有像素)。

H CompressARC 经验观察到的能力与局限性

图 12：谜题 28e73c20 及其 CompressARC 的解决方案。

CompressARC 可以执行的能力清单简述包括：

- **** 为单个过程分配特定颜色（参见谜题 0ca9ddb6）。
- **** 填充（Infilling）（参见谜题 0dfd9992）。
- **** 裁剪（Cropping）（参见谜题 1c786137）。
- **** 用线条连接点，包括 45 度对角线（参见谜题 1f876c06）。
- **** 同色检测（参见谜题 1f876c06）。
- **** 识别像素相邻关系（参见谜题 42a50994）。
- **** 为单个示例分配特定颜色（参见谜题 3bd67248）。
- **** 识别形状的组成部分（参见谜题 025d127b）。
- **** 短距离平移（参见谜题 025d127b）。

我们认为这些能力是由架构中特定的层赋予的，这些层是我们为了赋予 CompressARC 这些能力而专门设计的。

CompressARC 无法执行的能力清单简述包括：

- **** 将两种颜色相互对应（参见谜题 0d3d703e）。

- **** 多次串联重复一项操作（参见谜题 0a938d79）。

- **** 计数/数字（参见谜题 ce9e57f2）。

- **** 平移、旋转、反射、缩放、图像复制（参见谜题 0e206a2e, 5ad4f10b, 和 2bcee788）。

- **** 检测拓扑性质，如连通性（参见谜题 7b601669）。

- **** 规划、模拟智能体的行为（参见谜题 2dd70a9a）。

- **** 模式的长距离延伸（见上文图 12b 中的谜题 28e73c20）。

第 26 页

I 基准模型 (BASELINES)

表 1 中的 U-Net 基准模型是为了观察在受到与 CompressARC 相同约束下的标准方法表现。这些约束包括：避免推理前的任何训练，以及在推理期间仅使用测试谜题作为训练数据。 该基准模型的训练算法包括将每个输入网格喂入 U-Net，并使用 U-Net 输出对输出网格的像素颜色进行分类。 输入网格与输出网格形状不匹配的谜题将被跳过，并假定得分为零。 在 10,000 步训练的后半段中最常出现的两个输出网格被用作两个预测方案。 我们没有改变网格的宽度或高度以使 ARC-AGI 网格适应 U-Net。 U-Net 的 BatchNorm 被替换为 GroupNorm，如果激活网格太小而无法进一步池化，则跳过中间的池化/上采样层。 我们尝试过应用随机增强变换并反转变换，但由于在训练集上表现变差（无增强为 2.5%，有增强为 1%），我们放弃了这个想法。

J 谜题求解准确率表

关于整个数据集的数值准确率报告，请参见表 4 和表 5。

表 4：CompressARC 在训练集上的解题准确率，作为推理学习步数的函数，展示了不同允许猜测次数 (pass@n) 的结果。 官方基准测试以 2 次允许猜测为准报告。总训练集求解时间是在 NVIDIA RTX 4070 GPU 上按顺序解题记录的。

训练迭代	时间	Pass@1	Pass@2	Pass@5	Pass@10	Pass@100	Pass@100
100	6小时	1.00%	2.25%	3.50%	4.75%	6.75%	6.75%
200	13小时	11.50%	14.25%	16.50%	18.25%	23.25%	23.50%
500	32小时	23.00%	27.50%	31.50%	33.50%	39.25%	40.75%
1000	65小时	28.00%	31.75%	35.50%	37.75%	43.75%	46.50%
2000	130小时	30.25%	34.75%	38.25%	41.50%	48.50%	52.75%

K 如何改进我们的工作

在 CompressARC 发布时，有几个想法我们曾考虑过或尝试过，但由于各种原因未能奏效。

K.1 通过谜题间权重共享进行联合压缩 模板算法 1 为每个单独的谜题包含了一个硬编码的 θ 值。 我们或许可以通过在所有谜题之间共享同一个 θ 来进一步缩短模板程序长度，从而增强压缩效果并产生更多正确的解。 算法 2 需要进行相应修改。

是否需要我继续为您翻译接下来的第 27 页和第 28 页?

继续

这里是文档第 27 页和第 28 页的中文翻译：

第 27 页

表 5：CompressARC 在评估集上的解题准确率，报告方式与表 4 相同。

训练迭代	时间	Pass@1	Pass@2	Pass@5	Pass@10	Pass@100	Pass@100
100	7 h	0.75%	1.25%	2.50%	2.25%	3.00%	3.00%
200	14 h	5.00%	6.00%	7.00%	7.75%	12.00%	12.25%
500	34 h	13.50%	17.75%	19.25%	15.00%	20.50%	21.50%
1000	69 h	16.75%	19.25%	21.75%	23.00%	26.00%	28.75%
2000	138 h	18.50%	**20.00%**	24.25%	26.00%	31.25%	33.75%

- **策略一**：所有谜题使用相同的网络权重并并行训练。 每个谜题被分配一定的权重扰动，并受到某种约束（例如 LoRA）。
- **策略二**：为每个谜题学习一个高维向量（“谜题嵌入”），并学习一个从谜题嵌入到网络权重的线性映射。 该映射充当基础的超网络（hypernetwork）。
- 不幸的是，测试这些方案需要将算法 3 改为并行运行而非串行，这会减慢研究迭代速度，因此我们未作进一步探索。

K.2 用于形状复制任务的类卷积层 这种改进更针对 ARC-AGI，在 AGI 的通用性上意义较小。 许多 ARC-AGI 谜题涉及将形状从一处复制到另一处，而我们的网络缺乏此类操作的归纳偏置。 卷积正是能够实现多位置形状复制的操作。

- 引入卷积操作存在问题。 卷积往往会比稀疏信号更多地放大网格噪声，因此其归纳偏置对于形状复制并不理想。
- 我们尝试修改卷积的形式，采用了热带卷积（Tropical convolution），它在玩具谜题上效果良好，但在真实的训练谜题上表现不足，因此该想法被搁置。

$$(f * g)(x) = \max_{y} f(x-y) + g(y)$$

第 28 页

当重复卷积且部分网格翻转 180 度时，中心像素往往会产生高激活，因此有时必须将中心像素归零以保留信号。

K.3 用于缓解后验坍塌的 KL 底线 我们在测试中注意到，某些关键张量的 KL 散度在学习过程中降至零后便再也无法恢复，这类似于 VAE 中的后验坍塌（posterior collapse）现象。 当张量 KL 降为零，网络便停止学习使用其编码信息。 若人为在训练前期将 KL 维持在零以上，网络可能会学习利用该张量的信息。 虽然我们实现了一套机制，但效果不明显，未来可能需要探索不同的 KL 底线衰减计划。

K.4 正则化 在模板算法 1 中，我们没有压缩权重 θ 以减少其占据的比特数。 若对 θ 也进行压缩，会在损失函数中产生额外的 KL 项，在特定限制下简化为对 θ 的 L_2 正则化。 考虑到 θ 贡献的比特数巨大，忽略这一点确实有些鲁莽，改进此处可能会提升结果。

L 关于 ARC-AGI 基准测试的附加细节

- **隐藏规则**：每个谜题都有一个将输入映射到输出的隐藏规则。 400 个训练谜题比 400 个评估谜题简单。
- **核心主题**：
  - **物体性**：物体持久存在，不会无故出现或消失。
  - **目标导向性**：物体可以是具有意图和追求目标的“智能体”。
  - **数字与计数**：可以对物体进行计数或根据形状、移动等进行数学比较。
  - **几何与拓扑**：物体形状（矩形、三角形等）可进行镜像、旋转、平移、形变等。
- **人类表现**：普通人能解决 76.2% 的训练集，专家能解决 98.5%。
- **评分**：你有两次猜测输出网格的机会。 猜测必须包括网格宽高及所有像素颜色，完全正确才算成功。

- **评分环境**：比赛限制在 12 小时内完成，无互联网访问，在受限环境中运行。

您希望我继续为您翻译接下来的第 29 页和第 30 页吗?

继续

这里是文档第 29 页和第 30 页的中文翻译：

第 29 页

比赛使用隐藏的半私有评估集进行评分。我们报告的分数是在公开评估集上获得的，该集合与我们在开展这项工作时无法接触到的半私有评估集难度相同。

M 额外案例研究 下面，我们展示另外两个谜题，并剖析 CompressARC 对它们的解法。

M.1 案例研究：包围盒 (BOUNDING BOX) 谜题 `6d75e8bb` 属于训练集划分，见图 13。

图 13：包围盒：训练集中的谜题 6d75e8bb。

M.1.1 观察网络学习：包围盒 人类解决方案：我们首先意识到输入是红色和黑色的，输出也是红色和黑色的，但部分黑色像素被替换成了浅蓝色像素。我们看到红色形状保持不变。我们注意到浅蓝色方框包围着红色形状，且它是包含该红色形状的最小可能包围框。此时，我们将输入复制到答案网格，然后确定红色形状的水平和垂直范围，并将该范围内的所有非红色像素涂成浅蓝色。

CompressARC 解决方案：见表 6。

M.1.2 解决方案分析：包围盒 图 14 展示了 z 中每个张量包含的信息量。在训练过程中，除了三个张量外，所有张量的信息内容都降为零。在 600 到 1000 步之间，我们看到 `[example, height, width, channel]` 张量的信息内容大幅下降，而输出的答案没有变化。我们认为该张量最初被用于识别输入中的浅蓝色像素，但随后这些信息被网络的非线性部分记住了，并利用 `[example, height, channel]` 和 `[example, width, channel]` 作为位置编码。

图 15 显示了这些张量在解码层的平均输出，以查看其中存储了什么信息。

M.2 案例研究：中心十字 (CENTER CROSS) 谜题 `41e4d17e` 属于训练集划分，见图 16a。

第 30 页

表 6：CompressARC 学习“包围盒”谜题解法的过程。

学习步数	CompressARC 在做什么?	采样的解决方案预测
50	采样输出的平均值显示，输入中的浅蓝色像素在输出中通常被保留。然而，输入中的黑色像素被随意且随机地涂成了浅蓝色和红色。CompressARC 似乎并不知道彩色输入/输出像素位于某种包围框内，或者该包围框在输入和输出网格中是相同的。	样本 / 预测 1 (guess 1) / 样本平均值
100	采样输出的平均值显示红色像素被限制在一个包围着浅蓝色像素的虚构矩形内。CompressARC 似乎已经察觉到其他示例对输入和输出像素使用了一个共同的包围框，但并不完全确定边界在哪里以及输出框内填什么颜色。尽管如此，预测 2（第二常采样的输出）显示正确答案现在已经经常被采样到了。	样本 / 预测 1 / 样本平均值 / 预测 2
150	采样输出的平均值显示虚构包围框内几乎所有的像素都被涂成了红色。CompressARC 已经想出了答案，进一步的训练只是在精炼答案。	样本 / 预测 1 / 样本平均值 / 预测 2

您希望我继续为您翻译接下来的最后几页吗？

继续

这里是文档第 31 页至第 36 页（最后一部分）的中文翻译：

第 31 页

人类解决方案：我们首先注意到输入由浅蓝色背景上的蓝色“气泡”形状（实际上只是正方形，但蓝色让我们联想到气泡）组成，输出也是如此 。但在输出中，现在有从每个气泡中心发出的洋红色射线 。我们将输入复制到答案网格，然后从每个气泡的中心向每个基本方向画出洋红色射线，直到边缘 。此时提交答案会发现错误，我们

注意到在给定的演示中，蓝色气泡颜色是画在洋红色射线之上的，而我们把射线画在了气泡之上 。因此，我们拿起蓝色，将射线穿过气泡的每个点纠正回蓝色 。

CompressARC 解决方案：我们不展示其随时间演变的解，因为我们认为这没什么意思；相反我们将进行描述 。学习过程中其答案变化不大 。它开始于复制输入网格，在某个时间点，洋红色的行和列开始出现，并慢慢固定在正确位置 。CompressARC 从未错误地将射线画在气泡之上；它始终保持正确的顺序 。

M.2.1 解决方案分析：中心十字 图 17 展示了 z 中每个张量的信息量。

图 17：将训练期间的 KL 损失分解为多张量 z 中每个独立形状张量的贡献。

第 32 页（图 16）

图 16：中心十字：训练集中的谜题 41e4d17e。

第 33 页

图 18：训练期间损失成分的分解。

- **(a) [example, height, width, channel] 张量**。顶层主成分比次优成分强 2496 倍。**该张量编码了气泡的中心** 。在 KL 贡献图中可以看到该张量的信息量随时间减少，CompressARC 可能正处于消除十字形表示并用像素替换它的过程中，因为像素占用的比特更少 。

- **(b) [color, channel] 张量**。该张量仅用于区分谜题中各颜色的个体角色。

N 提到的 ARC-AGI-1 谜题列表

谜题 ID	说明
025d127b	涉及平移和识别形状部分
0a938d79	涉及多次串行重复操作
0ca9ddb6	涉及为特定过程分配颜色

谜题 ID	说明

第 34 页（谜题展示 1）

- **0d3d703e**: 颜色映射任务 。

- **0dfd9992**: 复杂背景下的填充 。

- **0e206a2e**: 涉及旋转/平移的对象变换 。

- **1c786137**: 噪声背景下的裁剪 。

- **1f876c06**: 连接点和同色检测 。

- **28e73c20**: 螺旋模式的长距离延伸 。

第 35 页（谜题展示 2）

- **272f95fa**: 本文案例研究中的“为方块着色” 。

- **2bcee788**: 涉及对象复制 。

- **2dd70a9a**: 复杂的路径规划/模拟 。

- **3bd67248**: 为每个示例分配特定颜色 。

- **41e4d17e**: 本文案例研究中的“中心十字” 。

- **42a50994**: 像素相邻关系识别 。

第 36 页

- **5ad4f10b**: 涉及形状收缩和颜色变更 。

- **6d75e8bb**: 本文案例研究中的“包围盒” 。

- **7b6016b9**: 涉及拓扑连通性检测 。

- **ce9e57f2**: 涉及数字与计数 。

表 7：提到的 ARC-AGI-1 谜题列表。所有这些谜题都是训练集划分的一部分。

O 代码 该项目的代码可在以下地址获取：
https://github.com/iliao2345/CompressARC

这份文档的翻译现已完成。如果您对 CompressARC 的**多张量通信层**设计、其**等变性机制**，或是它在特定谜题上的**学习曲线**有任何进一步的问题，我可以为您进行深入解答。

帮我找下GITHUB库

根据您上传的论文内容，该项目 **CompressARC** 的 GitHub 代码库地址为：

https://github.com/iliao2345/CompressARC

该地址出现在论文的最后一页（第 36 页）“O 代码 (O CODE)”部分。

arXiv:2512.06104v1 [cs.LG] 5 Dec 2025

ARC-AGI WITHOUT PRETRAINING

Isaac Liao[*]
Carnegie Mellon University
`iliao@andrew.cmu.edu`

Albert Gu
Carnegie Mellon University
`agu@andrew.cmu.edu`

ABSTRACT

Conventional wisdom in the age of LLMs dictates that solving IQ-test-like visual puzzles from the ARC-AGI-1 benchmark requires capabilities derived from massive pretraining. To counter this, we introduce *CompressARC*, a 76K parameter model without any pretraining that solves 20% of evaluation puzzles by minimizing the description length (MDL) of the target puzzle purely during inference time. The MDL endows CompressARC with extreme generalization abilities typically unheard of in deep learning. To our knowledge, CompressARC is the only deep learning method for ARC-AGI where training happens only on a single sample: the target inference puzzle itself, with the final solution information removed. Moreover, CompressARC does not train on the pre-provided ARC-AGI "training set". Under these extremely data-limited conditions, we do not ordinarily expect any puzzles to be solvable at all. Yet CompressARC still solves a diverse distribution of creative ARC-AGI puzzles, suggesting MDL to be an alternative feasible way to produce intelligence, besides conventional pretraining.

1 INTRODUCTION

The ARC-AGI-1 benchmark consists of abstract visual reasoning puzzles designed to evaluate a system's ability to rapidly acquire new skills from minimal input data (Chollet, 2019). Recent progress in LLM-based reasoning has shown impressive skill acquisition capabilities, but these systems still rely on massive amounts of pretraining data. In this paper, we explore how little data is truly required to tackle ARC-AGI by introducing *CompressARC*, a solution method derived from the Minimum Description Length (MDL) principle (Rissanen, 1978). CompressARC performs all of its learning at inference time and achieves 20% accuracy on ARC-AGI-1 evaluation puzzles—using only the puzzle being solved as input data.

The key to CompressARC's extreme data efficiency is its formulation as a code-golfing problem: to find the shortest possible self-contained program that outputs the entire ARC-AGI dataset, with any unsolved grids filled arbitrarily. By Occam's razor, we may expect the shortest program to fill the unsolved grids in the most sensible way, which is with "correct" solutions that match the rules of the puzzle. An overly basic program might store a hard-coded string of the puzzle data (plus arbitrary solutions) for output; but the program will be too long, implying the outputted solutions will be wrong. On the other hand, finding the optimally shortest program would require exhaustively enumerating many candidate programs, which is computationally infeasible (Solomonoff, 1964; Hutter, 2005). CompressARC strikes a new type of balance by overfitting a neural network to the puzzle data to compress the puzzles into weight matrices; these weights can be hard-coded into the program instead of the puzzles themselves, and then used within the program to recover the memorized puzzles with solutions. With careful counting of the bit length of the hard-coded weights, this technique converts the combinatorial search of finding the best program into a differentiable optimization problem, allowing us to minimize program length in a reasonable amount of time and still generate good solution predictions.

This framing preserves several attractive properties of the original code-golf formulation, all of which are novel when it comes to neural solutions to ARC-AGI:

- **No pretraining:** Since we begin with the target puzzle(s) in hand, no training phase is required.

---

[*] `https://iliao2345.github.io/`

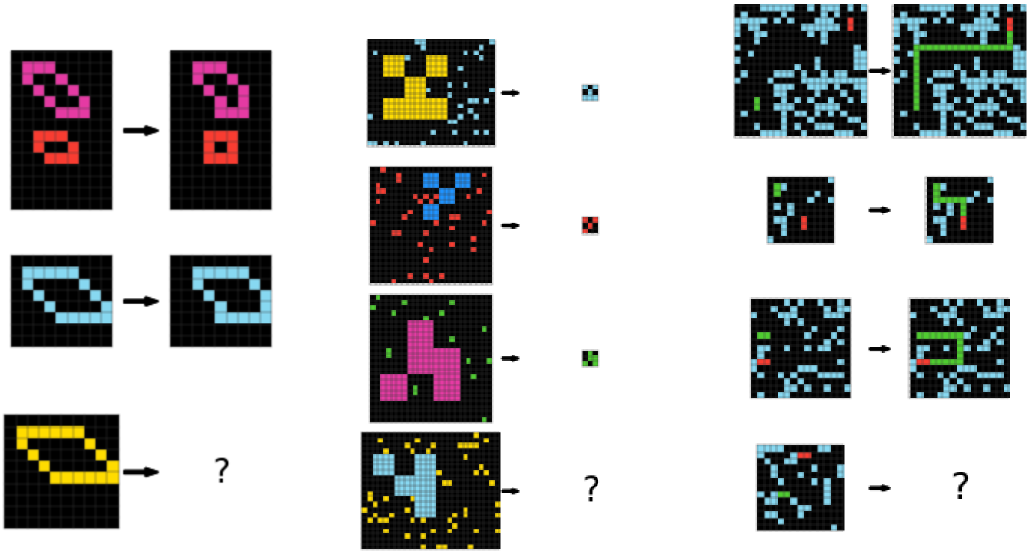
- **Inference-time learning:** Program length is minimized solely during inference by optimizing network weights with respect to the target puzzle(s).
- **Minimal data requirement:** Following Occam's razor, we assume strong generalization from the shortest program and use only the puzzle(s) themselves—no additional data is loaded into memory.

Despite never using the training set, performing no pretraining, and having only 76K parameters in its network, CompressARC generalizes to solve 20% of evaluation puzzles and 34.75% of training puzzles—performance that would be impossible for traditional deep learning methods under these constraints. CompressARC's strong performance in this setting suggests that bringing information compression to other data-limited contexts beyond ARC-AGI (e.g., drug discovery, protein design) may help us extract stronger capabilities in those applications as well.

The remainder of this paper introduces the ARC-AGI benchmark (Section 2), details the problem framing (Section 3), describes CompressARC's architecture (Section 4), presents empirical results (Section 5), and concludes with a discussion of implications (Section 6).

2 BACKGROUND: THE ARC-AGI BENCHMARK

ARC-AGI-1 is an artificial intelligence benchmark designed to test a system's ability to acquire new skills from minimal examples (Chollet, 2019). Each puzzle in the benchmark consists of a different hidden rule, which the system must apply to an input colored grid to produce a ground truth target colored grid. The hidden rules make use of themes like objectness, goal-directedness, numbers & counting, basic geometry, and topology. Several input-output grid pairs are given as examples to help the system figure out the hidden rule in the puzzle, and no other information is given. Figure 1 shows three examples of ARC-AGI-1 training puzzles.



(a) **Hidden rule:** Shift every object right by one pixel, except the bottom/right edges of the object.

(b) **Hidden rule:** Shrink the big object and set its color to the scattered dots' color.

(c) **Hidden rule:** Extend the green line to meet the red line by turning when hitting a wall.

Figure 1: Three example ARC-AGI-1 puzzles.

While solutions based on LLMs built using internet scale data have scored 87.5% on this benchmark (Chollet, 2024), and neural networks trained on only ARC-AGI data have scored 40.3% (Wang et al., 2025), CompressARC takes the data-minimal perspective to its limits, opting to only train on the test puzzle.

Appendix L contains more details about the ARC-AGI-1 benchmark. An extended survey of other related work, including other approaches for solving ARC-AGI, is also included in Appendix A. Note that we will generally refer to ARC-AGI-1 just as ARC-AGI in this paper.

3 METHOD

CompressARC tries to solve the problem of compressing the data into as short a program as possible, to obtain the puzzle solutions while keeping the program search feasible. In this case, the code must be entirely self-contained, receive no inputs, and must print out the entire ARC-AGI dataset of puzzles with any solutions filled in. It is typically infeasible to run a full algorithmic search to find the absolute shortest program, because we would have to search through a huge number of increasingly lengthy programs to find one whose printout matches our requirements. CompressARC makes program search more amenable by restricting itself to a suitably well-conditioned subspace of programs. Our overall strategy, which we will explain in more detail in the following sections, will be as follows:

- We define a space of programs through a template program (Algorithm 1) which contains numerous empty spaces for hardcoded values ("seeds") to be included. These hardcoded seeds will complete the program, so that it can be run, and the program will print out puzzles with solutions.
- We define a search algorithm (Algorithm 2) which enumerates many possible seeds to place in these spaces, and selects a good set of seeds to make Algorithm 1 as short as possible, with the constraint that its outputted puzzles are the given puzzles, but the solutions are unconstrained. Most of the algorithm length will be taken up by the length of the seeds, so it suffices to only count up the total length of the seeds to approximate the program length.
- Only some parts of Algorithm 2 are necessary to output the solution; the rest are only involved in generating the actual compressed representation of the puzzle. To solve the puzzles, we only keep the former parts, and we approximate parts of the search with a gradient descent procedure (Algorithm 3). Our differentiable approximations of the various seed lengths will be analogous to the individual components of the loss function for a variational autoencoder (VAE) (Kingma & Welling, 2022).

Puzzle/solution data format: Each puzzle takes the form of a tensor of shape [`n_example`, `width`, `height`, 2], containing color designations for every pixel in the $2 \times$ `n_example` grids. Here, `n_example` counts the total number of input/output grid pairs in the puzzle, *including the test grid pair whose output grid is not known*. The shapes listed in this section are for explanatory purposes and the actual data format is introduced in Section 4. A naive first try at code-golfing the dataset may involve writing a program that hard-codes each puzzle in a giant string and prints it out. Improvements can be made by clever ways of de-duplicating structures in the printed data (e.g., introducing **for** loops, etc.); we will detail our own particular strategy below.

3.1 RESTRICTING THE PROGRAM SPACE

We begin to derive CompressARC by picking a program subspace consisting of a template program (Algorithm 1) to be completed by substituting various hard-coded values into designated locations (shown in red). The template program generates each puzzle independently, and performs the operations for every puzzle:

1. It randomly samples a tensor z of shape [`n_example`, `n_colors`, `width`, `height`, 2] from a standard normal distribution, (line 4) (There will be a sampling seed, whose length will be analogous to a VAE's KL loss term.)
2. processes this with a neural network equivariant_NN which outputs a [`n_example`, `n_colors`, `width`, `height`, 2]-shaped logit tensor, (line 6)
3. and obtains a [`n_example`,`width`, `height`, 2]-shaped puzzle by sampling colors from the probability distribution implied by the logit tensor (line 8). We hope these colors match the given puzzle. (There will again be a sampling seed, whose length will be analogous to a VAE's reconstruction loss term.)

The neural network equivariant_NN is a neural network that is "equivariant" to valid transformations of ARC-AGI puzzles. This means when the input/output pair order or colors are permuted, or a flip or rotation is applied to the input z, then the output of the neural network will also be transformed in the same way.

Algorithm 1: Template for a short program that produce completed puzzles P_{filled} with solutions filled in. Red text is to be substituted in with hard-coded values produced via Algorithm 2.

```
1  Define an equivariant_NN architecture;
2
3  Set seed_z = <seed_z_1>;                          Hardcoded seed from Algo 2, puzzle 1
4  $z \leftarrow \text{sample}_{\text{seed\_z}}(N(0,1))$;                     Generate inputs z
5  Set $\theta = < \theta_1 >$;                          Hardcoded weights from Algo 2, puzzle 1
6  grid_logits $\leftarrow$ equivariant_NN$_\theta(z)$;                        Forward pass
7  Set seed_error = <seed_error_1>;                  Hardcoded seed from Algo 2, puzzle 1
8  $P_{\text{filled}} \leftarrow \text{sample}_{\text{seed\_error}}$(grid_logits);               Generate puzzle
9  Print $P_{\text{filled}}$
10
11 Set seed_z = <seed_z_2>;                          Hardcoded seed from Algo 2, puzzle 2
12 (...code repeats for all puzzles)                                                ...
```

The template program allows for two pseudo-random sampling seeds to be filled in for every puzzle (lines 3 and 7). The resulting printed puzzles can be guaranteed to match the true puzzles in the dataset by manipulating the final seed in line 7, which works after choosing any seed on line 3. With this guarantee in place, we can sum up the length of the code for the template, and find that the total length varies based on the number of bits/digits required to write down the two seeds. So, in order to search for short programs, we just need to make all the seeds as short as possible.

Multiple areas of the program can be adjusted to help minimize the seed length, and we will cover each in respective sections: the seeds and the weights on lines 3, 5, and 7 (Section 3.2 below), and the architecture on line 1 (Section 4).

3.2 SEED OPTIMIZATION

Algorithm 2 presents a method of optimizing the seeds and weights in template Algorithm 1 to reduce the total seed length. It first tries to manipulate the seed on line 3 of the template to imitate z being sampled from a different learned normal distribution (line 8), and then tries to manipulate the second seed to guarantee matching puzzle output (line 11). It then performs gradient descent on the normal distribution parameters and the neural network weights to minimize the total seed length (lines 13-14).

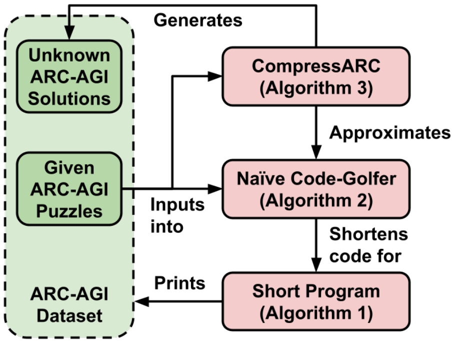

Generates
Unknown ARC-AGI Solutions
CompressARC (Algorithm 3)
Approximates
Given ARC-AGI Puzzles
Inputs into
Naïve Code-Golfer (Algorithm 2)
Shortens code for
ARC-AGI Dataset
Prints
Short Program (Algorithm 1)

Figure 2: CompressARC approximates a specific compression algorithm that converts the ARC-AGI puzzle dataset into the shortest program that prints it out exactly, along with any solutions. These printed solutions are assumed to be good predictors of the actual solutions, according to Occam's razor.

The idea of "manipulating the sampling seed for one distribution to imitate sampling from another distribution" tends to be fraught with technicalities and subtleties. Under favorable conditions, this is cleanly achievable through rejection sampling (Forsythe, 1972), which produces a sample from the imitated distribution using a seed whose expected length is the max log probability ratio between the two distributions. This can subsequently be improved and extended towards more general conditions using Relative Entropy Coding (REC) (Harsha et al., 2010; Havasi et al., 2018; Flamich et al., 2021), where the expected seed length lowers to the KL divergence between the two distributions, at the cost of only approximately achieving the desired sampling distribution. We refer readers to these references for details.

Our main issue with seed manipulation using REC within Algorithm 2 on lines 8 and 11 is that running REC requires exponential time in the divergence between the imitated distribution and the sampling distribution (Flamich et al., 2021). So to make it faster, we skip these steps and imitate their expected downstream consequences instead, resulting in CompressARC (Algorithm 3). Namely for

Algorithm 2: Minimize Description Length, a.k.a. code-golf. `n_example` denotes the total number of input/output pairs in the puzzle, *including the test pair where the output is unknown.* Line 11: The smallest possible seed_error is picked so that the sampled puzzle P_{filled} on line 12 matches the true puzzle P on both the input and output grids of demonstration pairs, as well as the test inputs, but with no restriction on the test outputs.

```
1  Input: ARC-AGI dataset;
2  Define an equivariant_NN architecture;
3  foreach puzzle $P$ in ARC-AGI dataset do
4      Randomly initialize weights $\theta$ for equivariant_NN$_\theta$;
5      Observe the dimensions n_example, n_colors, width, height of puzzle $P$;
6      Initialize input distribution: $\mu$ of shape [n_example, n_colors, width, height, 2],
         and diagonal $\Sigma$;
7      foreach step do
8          Set seed_z, by manipulating to imitate $z \sim N(\mu, \Sigma)$;
9          $z \leftarrow$ sample$_{\text{seed\_z}}(N(0, I)$, shape=[n_example, n_colors, width, height, 2]);
10         grid_logits $\leftarrow$ equivariant_NN$_\theta(z)$;
11         Set seed_error, by manipulating to obtain desired puzzle $P$;
12         $P_{\text{filled}} \leftarrow$ sample$_{\text{seed\_error}}$(grid_logits);
13         $L \leftarrow$ len(seed_z) $+$ len(seed_error);                    $\approx$len(Algo 1) + C
14         $\mu, \Sigma, \theta \leftarrow$ Adam$(\nabla_\mu L, \nabla_\Sigma L, \nabla_\theta L)$;
15     end foreach
16     Insert values seed_z, $\theta$, and seed_error into the pseudo-code for Algo 1;
17 end foreach
18 Return code for Algo 1
```

line 8, z is now directly sampled from the imitated distribution, and the seed length from line 13 is replaced by its expected length, which is very close to the KL divergence between the imitated and sampling distributions according to the properties of REC (Flamich et al., 2021). For line 11, the unknown grids of the puzzle are sampled directly, and the seed length from line 13 is approximated closely by the negative log likelihood of sampling the known grids exactly, i.e., the cross-entropy (see Appendix B for derivation).

Algorithm 3: CompressARC. It is the same as Algorithm 2, but with simulated seed manipulation, and truncated to return solved puzzles instead of description.

```
Input: ARC-AGI dataset;
Define an equivariant_NN architecture;
foreach puzzle $P$ in ARC-AGI dataset do
    Randomly initialize weights $\theta$ for equivariant_NN$_\theta$;
    Observe the dimensions n_example, n_colors, width, height of puzzle $P$;
    Initialize input distribution: $\mu$ of shape [n_example, n_colors, width, height, 2],
      and diagonal $\Sigma$;
    foreach step do
        $z \leftarrow$ sample$(N(\mu, \Sigma))$;
        grid_logits $\leftarrow$ equivariant_NN$_\theta(z)$;
        $L \leftarrow$ KL$(N(\mu, \Sigma) || N(0, I))$ $+$ cross-entropy(grid_logits, $P$);      $\approx$len(Algo 1) + C
        $\mu, \Sigma, \theta \leftarrow$ Adam$(\nabla_\mu L, \nabla_\Sigma L, \nabla_\theta L)$;
    end foreach
    $P_{\text{filled}} \leftarrow$ sample(grid_logits);
    Add $P_{\text{filled}}$ to solved puzzles
end foreach
Return solved puzzles
```

Algorithm 3 (CompressARC) is now able to automatically compress the ARC-AGI dataset through successive refinement of template Algorithm 1, outputting solutions afterward. The only remaining component to specify is the neural network architecture used within template Algorithm 1, which we will design by hand. Since the architecture definition only appears once in Algorithm 1 while seeds appear repeatedly for every puzzle, using a sophisticated architecture can help us shorten the length of the template Algorithm 1, by trading off architecture description length to allow for shorter seeds. This serves as the primary motivation for us to heavily engineer the neural network architecture.

4 ARCHITECTURE

The central idea in designing the neural network architecture is to create a high probability of sampling the ARC-AGI puzzles, consequently reducing the length of the seeds and by extension the length of template Algorithm 1. According to the template structure, this means we need the neural network to have good inductive biases for transforming noise into reasonable-seeming ARC-AGI puzzles. *The training puzzles play no role in our method other than to boost our efforts to better engineer the inductive biases into our layers.*

Since ARC-AGI puzzles would be just as likely to appear in any combination of input/output example orderings, colors, orientations, etc., we want our network to assign them all equal probability by default. So, we made our architecture equivariant to example permutations, color permutations, rotations, and flips; (Cohen & Welling, 2016a) guaranteeing that computations applied to transformed inputs result in equivalently transformed outputs. For any asymmetries a puzzle may have, we require Algorithm 3 to manipulate the seed of the random input z, to bias the outputted puzzle one way or another.

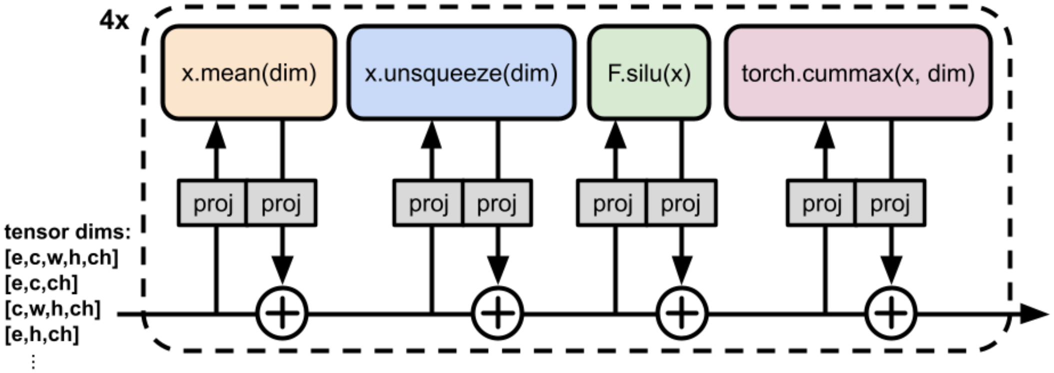


Figure 3: Core structure of CompressARC's neural network, which operates on multitensor data. Individual operations (colored) read and write to a residual backbone through learned projections (grey) in the `channel` dimension. The network is equivariant to permutations of indices along the other, non-`channel` dimensions as a result. Some layers like cummax break certain geometric symmetries, giving the architecture specific geometric abilities listed in Appendix H. Normalization, softmax, shift, and directional layers are not shown.

The architecture, shown in Figure 3, consists of a decoding layer functioning like an embedding matrix (details in Appendix D.1), followed by a core with a residual backbone, followed by a linear readout on the `channel` dimension (see Appendix D.8). In the core, linear projections on the `channel` dimension read data from the residual into specially designed operations, which write their outputs back into the residual through another projection. Normalization operations are scattered throughout the layers, and then the whole block of core layers is repeated 4 times. This is much like a transformer architecture, (Vaswani et al., 2023) except that the specially designed operations are not the attention and nonlinear operations on sequences, but instead the following operations on puzzle-shaped data:

- summing one tensor along an axis and/or broadcasting the result back up, (see Appendix D.2)
- taking the softmax along one or multiple axes of a tensor, (see Appendix D.3)
- taking a cumulative max along one of the geometric dimensions of a tensor, (see Appendix D.4)
- shifting by one pixel along one of the geometric dimensions of a tensor, (see Appendix D.4)

- elementwise nonlinearity, (see Appendix D.6)
- normalization along the `channel` dimension, (see Appendix D.7)

along with one more described in Appendix D.5. The operations have no parameters and have their behaviors controlled by their residual read/write weights. All of these read/write projections operate on the `channel` dimension. We used 16 channels in some parts of the backbone and 8 in others to reduce computational costs. Since these projections take up the majority of the model weights, the entire model only has 76K parameters.

The actual data format that the neural network uses for computation is not a single tensor shaped like [`n_example`, `n_colors`, `width`, `height`, `channel`], but instead a bucket of tensors that each have a different subset of these dimensions, for example a [`n_colors`, `width`, `channel`] tensor. Both the input z to the network and the outputted logits, as well as all of the internal activations, take the form of a multitensor. Generally, there is a tensor for every subset of these dimensions for storing information of that shape, which helps to build useful inductive biases. For example, an assignment of grid columns to colors can be stored as a one-hot table in the [`color`, `width`, `channel`]-shaped tensor. More details on multitensors are in Appendix C.

5 RESULTS

We gave CompressARC 2000 inference-time training steps on every puzzle, taking about 20 minutes per puzzle. CompressARC correctly solved 20% of evaluation set puzzles and 34.75% of training set puzzles within this budget of inference-time compute. Figure 4 shows the performance increase as more inference-time compute is given. Tables 4 and 5 in the Appendix document the numerical solve accuracies with timings.

Table 1: Comparison of solution accuracies of various methods for ARC-AGI-1, sorted by the amount of training data used. Each method makes two solution guesses per puzzle, and a guess is only correct if the grid shape and pixel colors are all correct. The U-Net (Ronneberger et al., 2015) baseline is a supervised model trained during inference time on only the demonstration input/output pairs of grids in the test puzzle to match the constraints of CompressARC; details in Appendix I.

Method	Trained on:	Neural	Acc.	Dataset split
Random guessing	Nothing	✗	0%	All
Brute force rule search (Kamradt, 2024)	Nothing	✗	40%	Private Eval
U-Net baseline	Target puzzle	✔	0.75%	Public Eval
CompressARC (ours)	**Target puzzle**	✔	**20%**	**Public Eval**
HRM ablation (ARC Prize Team, 2025)	Test puzzles	✔	31%	Public Eval
HRM (Wang et al., 2025)	Train+test puzzles	✔	40.3%	Public Eval
OpenAI o3 high (Chollet, 2024)	Internet scale data	✔	87.5%	Semi-Priv. Eval

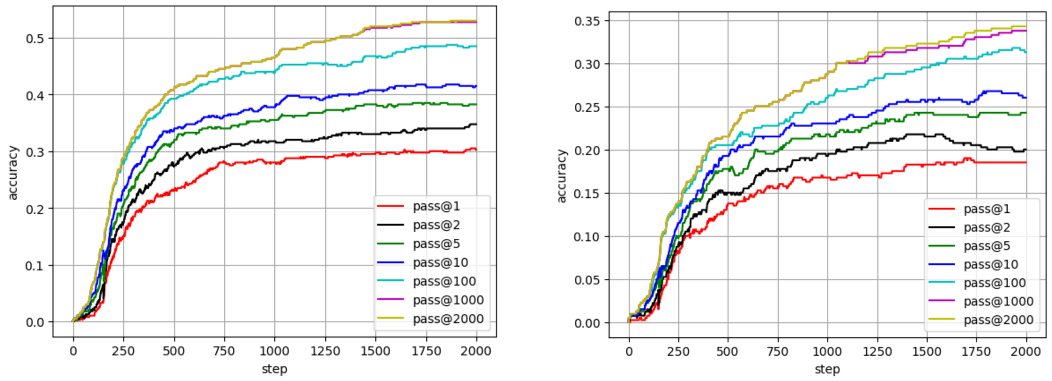
5.1 WHAT PUZZLES CAN AND CAN'T WE SOLVE?

CompressARC tries to use its abilities to figure out as much as it can, until it gets bottlenecked by one of it's inabilities.

For example, puzzle 28e73c20 in the training set requires extension of a pattern from the edge towards the middle, as shown in Figure 12a in the Appendix. Given the layers in it's network, CompressARC is generally able to extend patterns for short ranges but not long ranges. So, it does the best that it can, and correctly extends the pattern a short distance before guessing at what happens near the center (Figure 12b, Appendix). Appendix H includes a list of which abilities we have empirically seen CompressARC able to and not able to perform.

5.2 CASE STUDY: COLOR THE BOXES

In the puzzle shown (Figure 5), one must color the boxes depending on which side of the grid the box is on. We call this puzzle "Color the Boxes".



(a) Training set of 400 puzzles.

(b) Evaluation set of 400 puzzles.

Figure 4: CompressARC's puzzle solve accuracy rises as inference time learning progresses. Various numbers of allowed solution guesses (pass@n) for accuracy measurement are shown. The official benchmark is reported with 2 allowed guesses, which is why we report 20% on the evaluation set.

Human Solution: We first realize that the input is divided into boxes, and the boxes are still there in the output, but now they're colored. We then try to figure out which colors go in which boxes. First, we notice that the corners are always black. Then, we notice that the middle is always magenta. And after that, we notice that the color of the side boxes depends on which direction they are in: red for up, blue for down, green for right, and yellow for left. At this point, we copy the input over to the answer grid, then we color the middle box magenta, and then color the rest of the boxes according to their direction.

CompressARC Solution: Table 2 shows CompressARC's learning behavior over time. After CompressARC is done learning, we can deconstruct its learned z distribution to find that it codes for a color-direction correspondence table and row/column divider positions (Figure 7).

During training, the reconstruction error fell extremely quickly. It remained low on average, but would spike up every once in a while, causing the KL from z to bump upwards at these moments, as shown in Figure 6a.

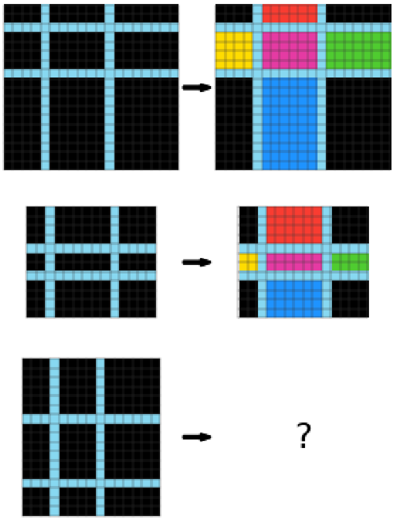


Figure 5: Color the Boxes, puzzle 272f95fa.

5.2.1 Solution Analysis

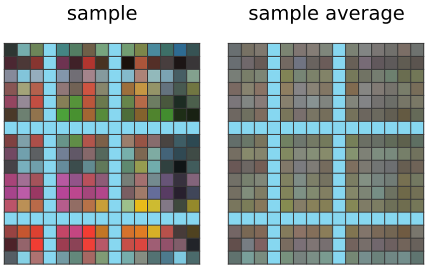
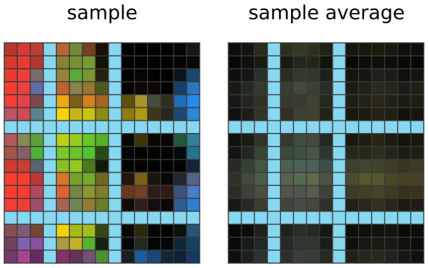
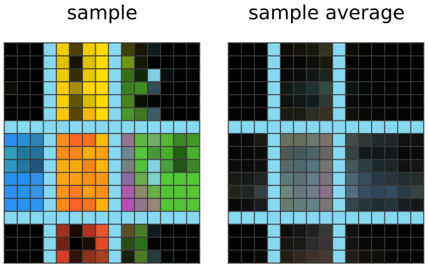
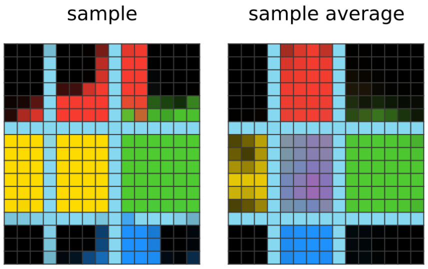
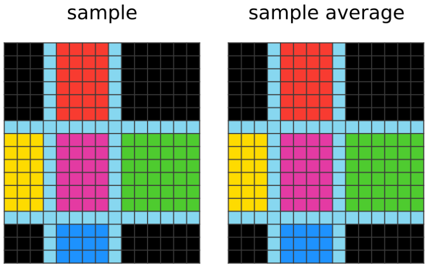
We observe the representations stored in z to see how CompressARC learns to solve Color the Boxes.

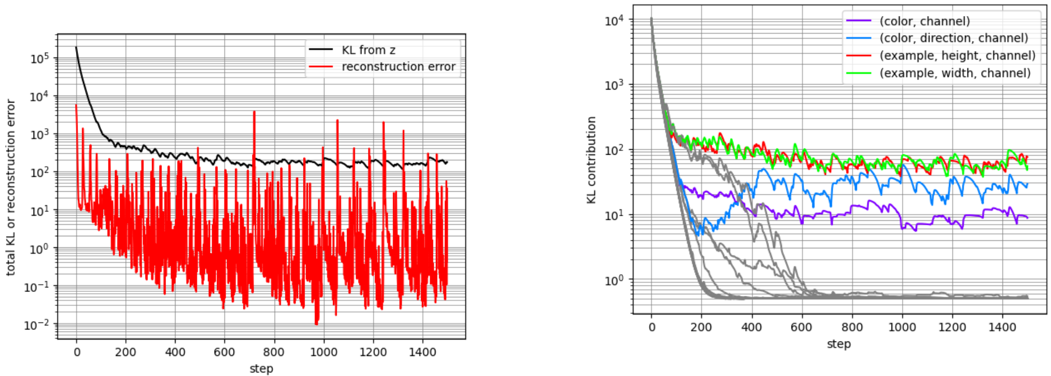
Since z is a multitensor, each of the tensors it contains produces an additive contribution to the total KL for z. By looking at the per-tensor contributions (see Figure 6b), we can determine which tensors in z code for information that is used to represent the puzzle.

All the tensors fall to zero information content during training, except for four tensors. In some replications of this experiment, we saw one of these four necessary tensors fall to zero information content, and CompressARC typically does not recover the correct answer after that. Here we are showing a lucky run where the [`color`, `direction`, `channel`] tensor almost falls but gets picked up 200 steps in, which is right around when the samples from the model begin to show the correct colors in the correct boxes.

We can look at the average output of the decoding layer (explained in Appendix D.1) corresponding to individual tensors of z, to see what information is stored there (see Figure 7). Each tensor contains a vector of dimension `n_channels` for various indices of the tensor. Taking the PCA of these

Table 2: CompressARC learning the solution for Color the Boxes, over time.

Learning steps	What is CompressARC doing?	Sampled solution guess
50	CompressARC's network outputs an answer grid (sample) with light blue rows/columns wherever the input has the same. It has noticed that all the other input-output pairs in the puzzle exhibit this correspondence. It doesn't know how the other output pixels are assigned colors; an exponential moving average of the network output (sample average) shows the network assigning mostly the same average color to non-light-blue pixels.	
150	The network outputs a grid where nearby pixels have similar colors. It has likely noticed that this is common among all the outputs, and is guessing that it applies to the answer too.	
200	The network output now shows larger blobs of colors that are cut off by the light blue borders. It has noticed the common usage of borders to demarcate blobs of colors in other outputs, and applies the same idea here. It has also noticed black corner blobs in other given outputs, which the network imitates.	
350	The network output now shows the correct colors assigned to boxes of the correct direction from the center. It has realized that a single color-to-direction mapping is used to pick the blob colors in the other given outputs, so it imitates this mapping. It is still not the best at coloring within the lines, and it is also confused about the center blob, probably because the middle does not correspond to a direction. Nevertheless, the average network output does show a tinge of the correct magenta color in the middle, meaning the network is catching on.	
1500	The network is as refined as it will ever be. Sometimes it will still make a mistake in the sample it outputs, but this uncommon and filtered out.	

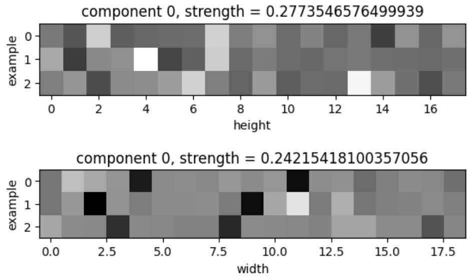


(a) Relative proportion of the KL and reconstruction terms to the loss during training, before taking the weighted sum. The KL dominates the loss and reconstruction is most often nearly perfect.

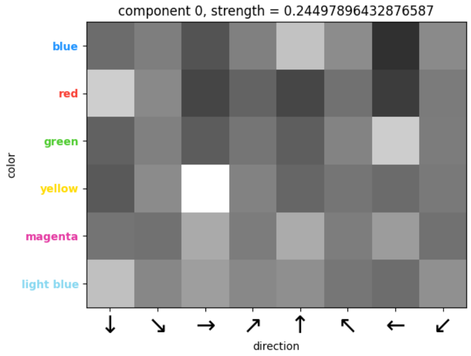
(b) Breaking down the KL loss during training into contributions from each individual shaped tensor in the multitensor z. Four tensors dominate, indicating they contain information, and the other 14 fall to zero, indicating their lack of information content.

Figure 6: Breaking down the loss components during training tells us where and how CompressARC prefers to store information relevant to solving a puzzle.

vectors reveals some number of activated components, telling us how many pieces of information are coded by the tensor.



(a) **`[example, height, channel]` and `[example, width, channel]` tensors.** For every example and row/column, there is a vector of dimension `n_channels`. Taking the PCA of this set of vectors, the top principal component (>1000 times stronger than the other components for both tensors) visualized as the [`example`, `height`] and [`example`, `width`] matrices shown above tells us which example/row and example/column combinations are uniquely identified by the stored information. **For every example, the two brightest pixels in the top matrix give positions of the light blue rows in the puzzle grids, and the darkest two pizels in the bottom matrix indicate the columns.**



(b) **`[direction, color, channel]` tensor.** A similar style PCA decomposition: the graph shows the top principal component for this tensor. The four brightest pixels identify blue with up, green with left, red with down, and yellow with right. **This tensor tells each direction which color to use for the opposite edge's box.** The top principal component is 829 times stronger than the next principal component.

Figure 7: Breaking down the loss components during training tells us where and how CompressARC prefers to store information relevant to solving a puzzle.

6 DISCUSSION

The prevailing reliance of modern deep learning on high-quality data has put the field in a chokehold when applied to problems requiring intelligent behavior that have less data available. This is especially true for the data-limited ARC-AGI benchmark, where LLMs trained on specially augmented, extended, and curated datasets dominate (Knoop, 2024). In the midst of this circumstance, we built CompressARC, which not only uses no training data at all, but forgoes the entire process of pretraining altogether. One should intuitively expect this to fail and solve no puzzles at all, but by applying MDL to the target puzzle during inference time, CompressARC solves a surprisingly large portion of ARC-AGI-1.

CompressARC's theoretical underpinnings come from minimizing the length of a programmatic description of the target puzzle. While other MDL search strategies have been scarce due to the intractablly large search space of possible programs, CompressARC explores a simplified, neural network-based search space through gradient descent. Though CompressARC's architecture is heavily engineered, its incredible ability to generalize from as low as two demonstration input/output pairs puts it in an entirely new regime of generalization for ARC-AGI.

Efficiency improvement remains a valuable direction for future work on CompressARC. CompressARC makes use of many custom operations (See Appendices C and D), and adding JIT-compiled kernels or fused CUDA kernels would increase the training iteration speed. Improvements will naturally have a larger effect on larger grids since our architecture's runtime scales with the number of pixels in the puzzle.

We challenge the assumption that intelligence must arise from massive pretraining and data, showing instead that clever use of MDL and compression principles can lead to surprising capabilities. We use CompressARC as a proof of concept to demonstrate that modern deep learning frameworks can be melded with MDL to create a possible alternative, complimentary route to AGI.

REFERENCES

Ekin Akyürek, Mehul Damani, Adam Zweiger, Linlu Qiu, Han Guo, Jyothish Pari, Yoon Kim, and Jacob Andreas. The surprising effectiveness of test-time training for few-shot learning. *arXiv preprint arXiv:2411.07279*, 2024.

ARC Prize Team. The hidden drivers of hrm's performance on arc-agi. `https://arcprize.org/blog/hrm-analysis`, August 2025. Accessed: 2025-11-24.

Guillermo Barbadillo. Solution summary for arc24. `https://ironbar.github.io/arc24/05_Solution_Summary/`, 2024. Accessed: 2025-05-12.

Guillermo Barbadillo. Exploring the combination of search and learn for the arc25 challenge, November 2025. URL `https://ironbar.github.io/arc25/05_Solution_Summary/#introduction-what-is-arc-and-why-is-it-relevant`. Accessed: 2025-11-22.

Jeremy Berman. How i got a record 53.6% on arc-agi, December 2024. URL `https://jeremyberman.substack.com/p/how-i-got-a-record-536-on-arc-agi`. Blog post on Substack.

Clément Bonnet and Matthew V Macfarlane. Searching latent program spaces, 2024. URL `https://arxiv.org/abs/2411.08706`.

J. Bretagnolle and C. Huber. Estimation des densités : Risque minimax. In C. Dellacherie, P. A. Meyer, and M. Weil (eds.), *Séminaire de Probabilités XII*, pp. 342–363, Berlin, Heidelberg, 1978. Springer Berlin Heidelberg. ISBN 978-3-540-35856-5.

Vinod Kumar Chauhan, Jiandong Zhou, Ping Lu, Soheila Molaei, and David A. Clifton. A brief review of hypernetworks in deep learning. *Artificial Intelligence Review*, 57(9), August 2024. ISSN 1573-7462. doi: 10.1007/s10462-024-10862-8. URL `http://dx.doi.org/10.1007/s10462-024-10862-8`.

François Chollet. On the measure of intelligence, 2019. URL `https://arxiv.org/abs/1911.01547`.

François Chollet. Openai o3 breakthrough high score on arc-agi-pub. `https://arcprize.org/blog/oai-o3-pub-breakthrough`, 2024. Accessed: 2025-05-12.

Taco Cohen and Max Welling. Group equivariant convolutional networks. In Maria Florina Balcan and Kilian Q. Weinberger (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 2990–2999, New York, New York, USA, 20–22 Jun 2016a. PMLR. URL `https://proceedings.mlr.press/v48/cohenc16.html`.

Taco S. Cohen and Max Welling. Group equivariant convolutional networks, 2016b. URL `https://arxiv.org/abs/1602.07576`.

Jack Cole and Mohamed Osman. Don't throw the baby out with the bathwater: How and why deep learning for arc. *arXiv preprint arXiv:2506.14276*, 2025.

Shiqing Fan, Liu Liying, and Ye Luo. An alternative practice of tropical convolution to traditional convolutional neural networks, 2021. URL `https://arxiv.org/abs/2103.02096`.

Gergely Flamich, Marton Havasi, and José Miguel Hernández-Lobato. Compressing images by encoding their latent representations with relative entropy coding, 2021. URL `https://arxiv.org/abs/2010.01185`.

George E. Forsythe. Von neumann's comparison method for random sampling from the normal and other distributions. *Mathematics of Computation*, 26(120):817–826, 1972. ISSN 00255718, 10886842. URL `http://www.jstor.org/stable/2005864`.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines, 2014. URL `https://arxiv.org/abs/1410.5401`.

Ryan Greenblatt. Getting 50% (sota) on arc-agi with gpt-4o. `https://redwoodresearch.substack.com/p/getting-50-sota-on-arc-agi-with-gpt`, 2024. Accessed: 2025-05-12.

Prahladh Harsha, Rahul Jain, David McAllester, and Jaikumar Radhakrishnan. The communication complexity of correlation. *IEEE Transactions on Information Theory*, 56(1):438–449, 2010. doi: 10.1109/TIT.2009.2034824.

Marton Havasi, Robert Peharz, and José Miguel Hernández-Lobato. Minimal random code learning: Getting bits back from compressed model parameters. *arXiv preprint arXiv:1810.00440*, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. URL `https://arxiv.org/abs/1512.03385`.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2023. URL `https://arxiv.org/abs/1606.08415`.

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-VAE: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, 2017. URL `https://openreview.net/forum?id=Sy2fzU9gl`.

Michael Hodel. Domain specific language for the abstraction and reasoning corpus. `https://github.com/michaelhodel/arc-dsl/blob/main/arc_dsl_writeup.pdf`, 2024. Accessed: 2025-05-12.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL `https://arxiv.org/abs/2106.09685`.

Marcus Hutter. *The Universal Algorithmic Agent AIXI*, pp. 141–183. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. ISBN 978-3-540-26877-2. doi: 10.1007/3-540-26877-4_5. URL `https://doi.org/10.1007/3-540-26877-4_5`.

Marcus Hutter. Hutter prize for lossless compression of human knowledge. `https://prize.hutter1.net/`, 2006. Accessed: 2025-05-12.

Greg Kamradt. Arc prize 2024 solution: 4th place score 40. Kaggle Notebook, 2024. URL `https://www.kaggle.com/code/gregkamradt/arc-prize-2024-solution-4th-place-score-40-811b72`. Accessed: 2025-11-22.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022. URL `https://arxiv.org/abs/1312.6114`.

Mike Knoop. ARC Prize 2024 Winners & Technical Report Published — arcprize.org. `https://arcprize.org/blog/arc-prize-2024-winners-technical-report`, 2024. [Accessed 12-05-2025].

A.N. Kolmogorov. On tables of random numbers. *Theoretical Computer Science*, 207(2):387–395, 1998. ISSN 0304-3975. doi: https://doi.org/10.1016/S0304-3975(98)00075-9. URL `https://www.sciencedirect.com/science/article/pii/S0304397598000759`.

G. G. Langdon. An introduction to arithmetic coding. *IBM Journal of Research and Development*, 28(2):135–149, 1984. doi: 10.1147/rd.282.0135.

Solim LeGris, Wai Keen Vong, Brenden M. Lake, and Todd M. Gureckis. H-arc: A robust estimate of human performance on the abstraction and reasoning corpus benchmark, 2024. URL `https://arxiv.org/abs/2409.01374`.

Wen-Ding Li, Keya Hu, Carter Larsen, Yuqing Wu, Simon Alford, Caleb Woo, Spencer M. Dunn, Hao Tang, Michelangelo Naim, Dat Nguyen, Wei-Long Zheng, Zenna Tavares, Yewen Pu, and Kevin Ellis. Combining induction and transduction for abstract reasoning, 2024a. URL `https://arxiv.org/abs/2411.02272`.

Wen-Ding Li, Keya Hu, Carter Larsen, Yuqing Wu, Simon Alford, Caleb Woo, Spencer M Dunn, Hao Tang, Michelangelo Naim, Dat Nguyen, et al. Combining induction and transduction for abstract reasoning. *arXiv preprint arXiv:2411.02272*, 2024b.

Victor Vikram Odouard. Arc-solution_documentation. `https://github.com/victorvikram/ARC-icecuber/blob/master/ARC-solution_documentation.pdf`, 2024. Accessed: 2025-05-12.

J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978. ISSN 0005-1098. doi: https://doi.org/10.1016/0005-1098(78)90005-5. URL `https://www.sciencedirect.com/science/article/pii/0005109878900055`.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241. Springer, 2015.

C. Shannon. The zero error capacity of a noisy channel. *IRE Transactions on Information Theory*, 2 (3):8–19, 1956. doi: 10.1109/TIT.1956.1056798.

C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3): 379–423, 1948. doi: 10.1002/j.1538-7305.1948.tb01338.x.

R.J. Solomonoff. A formal theory of inductive inference. part i. *Information and Control*, 7(1):1–22, 1964. ISSN 0019-9958. doi: https://doi.org/10.1016/S0019-9958(64)90223-2. URL `https://www.sciencedirect.com/science/article/pii/S0019995864902232`.

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei A. Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts, 2020. URL `https://arxiv.org/abs/1909.13231`.

Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. Ladder variational autoencoders, 2016. URL `https://arxiv.org/abs/1602.02282`.

Alexandre B. Tsybakov. *Introduction to Nonparametric Estimation*. Springer Publishing Company, Incorporated, 1st edition, 2008. ISBN 0387790519.

Arash Vahdat and Jan Kautz. Nvae: A deep hierarchical variational autoencoder, 2021. URL `https://arxiv.org/abs/2007.03898`.

Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning, 2018. URL `https://arxiv.org/abs/1711.00937`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL `https://arxiv.org/abs/1706.03762`.

Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi Yadkori. Hierarchical reasoning model. *arXiv preprint arXiv:2506.21734*, 2025.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On layer normalization in the transformer architecture, 2020. URL `https://arxiv.org/abs/2002.04745`.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander Smola. Deep sets, 2018. URL `https://arxiv.org/abs/1703.06114`.

A RELATED WORK

A.1 EQUIVALENCE OF COMPRESSION AND INTELLIGENCE

The original inspiration of this work came from the Hutter Prize (Hutter, 2006), which awards a prize for those who can compress a file of Wikipedia text the most, as a motivation for researchers to build intelligent systems. It is premised upon the idea that the ability to compress information is equivalent to intelligence.

This equivalence between intelligence and compression has a long history. For example, when talking about intelligent solutions to prediction problems, the ideal predictor implements Solomonoff Induction, a theoretically best possible but uncomputable prediction algorithm that works universally for all prediction tasks (Solomonoff, 1964). This prediction algorithm is then equivalent to a best possible compression algorithm whose compressed code length is the Kolmogorov Complexity of the data (Kolmogorov, 1998). This prediction algorithm can also be used to decode a description of the data of minimal length, linking these formulations of intelligence to MDL (Rissanen, 1978). In our work, we try to approximate this best possible compression algorithm with a neural network.

A.2 INFORMATION THEORY AND CODING THEORY

Since we build an information compression system, we make use of many results in information theory and coding theory. The main result required to motivate our model architecture is the existence of Relative Entropy Coding (REC) (Flamich et al., 2021). The fact that REC exists means that as long as a KL divergence can be bounded, the construction of a compression algorithm is always possible and the issue of realizing the algorithm can be abstracted away. Thus, problems about coding theory and translating information from Gaussians into binary and back can be ignored, since we can figure out the binary code length directly from the Gaussians instead. In other words, we only need to do enough information theory using the Gaussians to get the job done, with no coding theory at all. While the existence of arithmetic coding (Langdon, 1984) would suffice to abstract the problem away when distributions are discrete, neural networks operate in a continuous space so we need REC instead.

Our architecture sends z information through an additive white Gaussian noise (AWGN) channel, so the AWGN channel capacity formula (Gaussian input Gaussian noise) plays a heavy role in the design of our decoding layer (Shannon, 1948).

A.3 VARIATIONAL AUTOENCODERS

The decoder side of the variational autoencoder (Kingma & Welling, 2022) serves as our decompression algorithm. While we would use something that has more general capabilities like a neural Turing machine (Graves et al., 2014) instead, neural Turing machines are not very amenable to gradient descent-based optimization so we stuck with the VAE.

VAEs have a long history of developments that are relevant to our work. At one point, we tried using multiple decoding layers to make a hierarchical VAE decoder (Sønderby et al., 2016) instead. This does not affect the KL calculation because a channel capacity with feedback is equal to the channel capacity without feedback (Shannon, 1956). But, we found empirically that the first decoding layer would absorb all of the KL contribution, making the later decoding layers useless. Thus, we only used one decoding layer at the beginning.

The beta-VAE (Higgins et al., 2017) introduces a reweighting of the reconstruction loss to be stronger than the KL loss, and we found that to work well in our case. The NVAE applies a non-constant weighting to loss components (Vahdat & Kautz, 2021). A rudimentary form of scheduled loss recombination is used in CompressARC.

A.4 OTHER ARC-AGI METHODS

Top-scoring methods to solve ARC-AGI rely on converting puzzle grids into text and then feeding them into a pretrained large language model which is prompted to find the solution. The predominant techniques involve either using the LLM to output the solution grid directly (Li et al., 2024b; Cole

& Osman, 2025; Akyürek et al., 2024), or output a program that can be run to manipulate the grids instead(Li et al., 2024b; Greenblatt, 2024; Barbadillo, 2025; Berman, 2024). Oftentimes, these methods employ several tricks to improve performance:

- Fine-tuning on training puzzle data
  - Applying data augmentation to increase the effective number of puzzles to fine-tune on (Akyürek et al., 2024)
  - Fine-tuning on synthetic data (Li et al., 2024a; Akyürek et al., 2024)
- Employing inference-time training approaches
  - Fine-tuning an individual model specific to each test puzzle, during test time (Akyürek et al., 2024)
  - Test-time training (TTT) techniques (Sun et al., 2020; Barbadillo, 2024)
- Sampling many model outputs or random augmentations of the test puzzle, for ensembling (Cole & Osman, 2025; Greenblatt, 2024)
- LLM reasoning (Chollet, 2024)

Such methods have managed to score up to 87.5% on the semi-private split of ARC-AGI, at a cost of over $200 equivalent of inference-time compute per puzzle (Chollet, 2024). These approaches all make use of language models that were pretrained on the entire internet, which is in contrast to CompressARC, whose only training data is the test puzzle. Their commonalities with CompressARC are mainly in the emphasis on training individual models on individual test puzzles and the use of ensembling to improve solution predictions.

Aside from these methods, several other methods have been studied:

- An older class of methods consists of hard-coded, large-scale searches through program spaces in hand-written domain-specific languages designed specifically for ARC (Hodel, 2024; Odouard, 2024). While these methods do not use neural networks to solve puzzles and are less instructive towards the field of machine learning, they share the commonality of using heavily engineered components designed specifically for ARC-AGI.
- (Bonnet & Macfarlane, 2024) introduced a VAE-based method for searching through a latent space of programs. This is the most similar work to ours that we found due to their VAE setup.

A.5 DEEP LEARNING ARCHITECTURES

We designed our own neural network architecture from scratch, but not without borrowing crucial design principles from many others.

Our architecture is fundamentally structured like a transformer, consisting of a residual stream where representations are stored and operated upon, followed by a linear head (Vaswani et al., 2023; He et al., 2015). Pre-and post-norms with linear up- and down-projections allow layers to read and write to the residual stream (Xiong et al., 2020). The SiLU-based nonlinear layer is especially similar to a transformer's (Hendrycks & Gimpel, 2023).

Our equivariance structures are inspired by permutation-invariant neural networks, which are a type of equivariant neural network (Zaheer et al., 2018; Cohen & Welling, 2016b). Equivariance transformations are taken from common augmentations to ARC-AGI puzzles.

B SEED LENGTH ESTIMATION BY KL AND CROSS-ENTROPY

In Section 3.1, we estimate len(seed_z) in line 12 of template Algorithm 2 as $\text{KL}(N(\mu, \Sigma)||N(0, I))$, and len(seed_error) as $\text{cross-entropy}(\text{grid\_logits}, P)$. In this section, we will argue for the reasonability of this approximation. Readers may also refer to Flamich et al. (2021), which introduces a better seed manipulation method as "Relative Entropy Coding" (REC). Flamich et al. (2021) shows that seed communication is effectively the most bit-efficient way for an encoder and decoder to communicate samples from a distribution Q if there is a shared source of randomness P. This uses nearly $\text{KL}(P||Q)$ expected bits per sample communicated. We urge readers to refer to Flamich et al. (2021) for details regarding the manipulation procedure, runtime and memory analysis, and approximation strength. Below, we follow with our own effort at reasoning through why.

To recap, we original procedure in Algorithm 2 manipulates the seed for sampling $z \sim N(0, I)$ to simulate as though $z \sim N(\mu, \Sigma)$, and we would like to show that we can closely approximate this sampling using an expected number of seed bits close to $\mathrm{KL}(N(\mu, \Sigma)||N(0, I))$.

For sake of illustration, suppose for instance that Algorithm 2 implements something similar to rejection sampling, (Forsythe, 1972) iterating through seeds one by one and accepting the sample with probability $\min(1, cw(z))$ for some $c \ll 1$, where $w(z)$ is the probability ratio

$$w(z) = \frac{N(z; \mu, \Sigma)}{N(z; 0, I)}$$

When we pick a small enough c, the sampling distribution becomes arbitrarily close to $N(\mu, \Sigma)$ as desired. With this c, we would like to show that the expected number of rejections leads us to end up with a seed length close to the KL.

We would first like to lower bound the probability P_{accept} of accepting at each step, which is

$$\begin{aligned} P_{\text{accept}} &= \int N(z; 0, I) \min(1, cw(z))\, \mathrm{d}z \\ &= \int N(z; 0, I) \min\left(1, \frac{cN(z; \mu, \Sigma)}{N(z; 0, I)}\right) \mathrm{d}z \\ &= \int \min\left(N(z; 0, I), cN(z; \mu, \Sigma)\right) \mathrm{d}z \end{aligned}$$

We will follow a modified version of a derivation of the Bretagnolle–Huber inequality (Bretagnolle & Huber, 1978) by Tsybakov (2008) to derive a bound on the KL:

$$\begin{aligned} (1+c)P_{\text{accept}} \geq & (1 + c - P_{\text{accept}})P_{\text{accept}} \\ &= \left(\int \max\left(N(z; 0, I), cN(z; \mu, \Sigma)\right) \mathrm{d}z\right) \left(\int \min\left(N(z; 0, I), cN(z; \mu, \Sigma)\right) \mathrm{d}z\right) \end{aligned}$$

where applying the Cauchy-Schwarz inequality with a function space inner product,

$$\begin{aligned} &\geq \left(\int \sqrt{\max\left(N(z; 0, I), cN(z; \mu, \Sigma)\right) \min\left(N(z; 0, I), cN(z; \mu, \Sigma)\right)}\, \mathrm{d}z\right)^2 \\ &= \left(\int \sqrt{cN(z; 0, I) N(z; \mu, \Sigma)}\, \mathrm{d}z\right)^2 \\ &= c \exp\left(2 \ln \int \sqrt{N(z; 0, I) N(z; \mu, \Sigma)}\, \mathrm{d}z\right) \\ &= c \exp\left(2 \ln \int N(z; \mu, \Sigma) \sqrt{\frac{N(z; 0, I)}{N(z; \mu, \Sigma)}}\, \mathrm{d}z\right) \\ &= c \exp\left(2 \ln \mathbb{E}_{z \sim N(z; \mu, \Sigma)} \left[\sqrt{\frac{N(z; 0, I)}{N(z; \mu, \Sigma)}}\right]\right) \end{aligned}$$

and following with Jensen's inequality,

$$\begin{aligned} &\geq c \exp\left(\mathbb{E}_{z \sim N(z; \mu, \Sigma)} \left[\ln \frac{N(z; 0, I)}{N(z; \mu, \Sigma)}\right]\right) \\ &= c \exp\left(-\mathrm{KL}(N(\mu, \Sigma)||N(0, I))\right) \end{aligned}$$

leads to an acceptance probability of at least

$$P_{\text{accept}} \geq \frac{c}{1+c} \exp\left(-\mathrm{KL}(N(\mu, \Sigma)||N(0, I))\right)$$

Therefore, according to the rejection sampling procedure, the number of samples proposed (i.e. the expected seed) is at most the inverse of this acceptance probability,

$$\text{seed\_z} \leq \frac{1+c}{c} \exp(\text{KL}(N(\mu, \Sigma)||N(0,1)))$$

so the expected seed length is at most around the logarithm,

$$\text{len(seed\_z)} \leq \text{KL}(N(\mu, \Sigma)||N(0,1)) + \log(1+c) - \log c$$

matching up with our stated KL approximation of the seed length.

For the seed_error term, Algorithm 2 manipulates the seed to sample a puzzle P from a distribution implied by some logits. This is effectively the same as sampling grid_logits $\sim$ Categorical_distribution(logits) and manipulating the seed to try to get grid_logits $\sim$ Delta_distribution(P). Then, the same KL-based bound on required seed length can be used once again. The expected seed_error length is at most

$$\text{KL}(\text{Delta\_distribution}(P)||\text{Categorical\_distribution}(\text{logits})) + \log(1+c) - \log c$$

which simplifies as

$$\begin{aligned}
&\mathbb{E}_{x\sim\text{Delta\_distribution}(P)}\left[\log \frac{\delta(x=P)}{\text{Categorical\_probability}(x;\text{logits})}\right] + \log(1+c) - \log c \\
&= \log \frac{\delta(P=P)}{\text{Categorical\_probability}(P;\text{logits})} + \log(1+c) - \log c \\
&= -\log \text{Categorical\_probability}(P;\text{logits}) + \log(1+c) - \log c \\
&= \text{cross\_entropy}(\text{logits}, P) + \log(1+c) - \log c
\end{aligned}$$

where the δ is 1 when the statement within is true, and 0 otherwise.

C MULTITENSORS

The actual data (z, hidden activations, and puzzles) passing through our layers comes in a format that we call a "**multitensor**", which is just a bucket of tensors of various shapes, as shown in Figure 8. All the equivariances we use can be described in terms of how they change a multitensor.

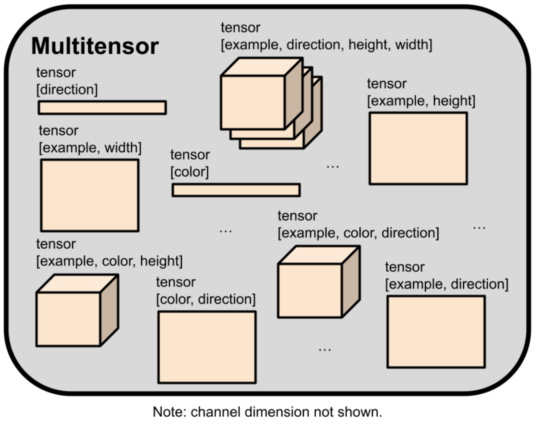

Multitensor
tensor [direction]
tensor [example, direction, height, width]
tensor [example, height]
tensor [example, width]
tensor [color]
...
tensor [example, color, direction]
tensor [example, color, height]
tensor [color, direction]
tensor [example, direction]
Note: channel dimension not shown.

Figure 8: Our neural network's internal representations come in the form of a "multitensor", a bucket of tensors of different shapes. One of the tensors is shaped like [`example`, `color`, `height`, `width`, `channel`], an adequate shape for storing a whole ARC-AGI puzzle.

Most common classes of machine learning architectures operate on a single type of tensor with constant rank. LLMs operate on rank-3 tensors of shape [`n_batch`, `n_tokens`, `n_channels`], and Convolutional Neural Networks (CNNs) operate on rank-4 tensors of shape [`n_batch`, `n_channels`, `height`, `width`]. Our multitensors are a set of varying-rank tensors of unique type, whose dimensions are a subset of a rank-6 tensor of shape [`n_example`, `n_colors`, `n_directions`, `height`, `width`, `n_channels`], as illustrated in Figure 8. We always keep the channel dimension, so there are at most 32 tensors in each multitensor. We also maintain several rules (see Appendix E.1) that determine whether a tensor shape is "legal" or not, which reduces the number of tensors in a multitensor to 18.

Dimension	**Description**
Example	Number of examples in the ARC-AGI puzzle, including the one with held-out answer
Color	Number of unique colors in the ARC-AGI puzzle, not including black, see Appendix F.2
Direction	8
Height	Determined when preprocessing the puzzle, see Appendix F.1
Width	Determined when preprocessing the puzzle, see Appendix F.1
Channel	In the residual connections, the size is 8 if the `direction` dimension is included, else 16. Within layers it is layer-dependent.

Table 3: Size conventions for multitensor dimensions.

To give an idea of how a multitensor stores data, an ARC-AGI puzzle can be represented by using the [`example`, `color`, `height`, `width`, `channel`] tensor, by using the `channel` dimension to select either the input or output grid, and the `height`/`width` dimensions for pixel location, a one hot vector in the `color` dimension, specifying what color that pixel is. The [`example`, `height`, `channel`] and [`example`, `width`, `channel`] tensors can similarly be used to store masks representing grid shapes for every example for every input/output grid. All those tensors are included in a single multitensor that is computed by the network just before the final linear head (described in Appendix D.8).

When we apply an operation on a multitensor, we by default assume that all non-`channel` dimensions are treated identically as batch dimensions by default. The operation is copied across the indices of dimensions unless specified. This ensures that we keep all our symmetries intact until we use a specific layer meant to break a specific symmetry.

A final note on the `channel` dimension: usually when talking about a tensor's shape, we will not even mention the `channel` dimension as it is included by default.

D LAYERS IN THE ARCHITECTURE

D.1 DECODING LAYER

This layer's job is to sample a multitensor z and bound its information content, before it is passed to the next layer. This layer and outputs the KL divergence between the learned z distribution and $N(0, I)$. Penalizing the KL prevents CompressARC from learning a distribution for z that memorizes the ARC-AGI puzzle in an uncompressed fashion, and forces CompressARC to represent the puzzle more succinctly. Specifically, it forces the network to spend more bits on the KL whenever it uses z to break a symmetry, and the larger the symmetry group broken, the more bits it spends.

This layer takes as input:

- A learned target multiscalar, called the "target capacity".[1] The decoding layer will output z whose information content per tensor is close to the target capacity,[2]
- learned per-element means for z,[3]
- learned per-element capacity adjustments for z.

---

[1] Target capacities are exponentially parameterized and rescaled by 10x to increase sensitivity to learning, initialized at a constant 10^4 nats per tensor, and forced to be above a minimum value of half a nat.

[2] The actual information content, which the layer computes later on, will be slightly different because of the per-element capacity adjustments.

[3] Means are initialized using normal distribution of variance 10^{-4}.

We begin by normalizing the learned per-element means for z.[4] Then, we figure out how much Gaussian noise we must add into every tensor to make the AWGN channel capacity (Shannon, 1948) equal to the target capacity for every tensor (including per-element capacity adjustments). We apply the noise to sample z, keeping unit variance of z by rescaling.[5]

We compute the information content of z as the KL divergence between the distribution of this sample and $N(0, 1)$.

Finally, we postprocess the noisy z by scaling it by the sigmoid of the signal-to-noise ratio.[6] This ensures that z is kept as-is when its variance consists mostly of useful information and it is nearly zero when its variance consists mostly of noise. All this is done 4 times to make a channel dimension of 4. Then we apply a projection (with different weights per tensor in the multitensor, i.e., per-tensor projections) mapping the channel dimension up to the dimension of the residual stream.

D.2 MULTITENSOR COMMUNICATION LAYER

This layer allows different tensors in a multitensor to interact with each other.

First, the input from the residual stream passes through per-tensor projections to a fixed size (8 for downwards communication and 16 for upwards communication). Then a message is sent to every other tensor that has at least the same dimensions for upwards communication, or at most the same dimensions for downwards communication. This message is created by either taking means along dimensions to remove them, or unsqueezing+broadcasting dimensions to add them, as in Figure 9. All the messages received by every tensor are summed together and normalization is applied. This result gets up-projected back and then added to the residual stream.

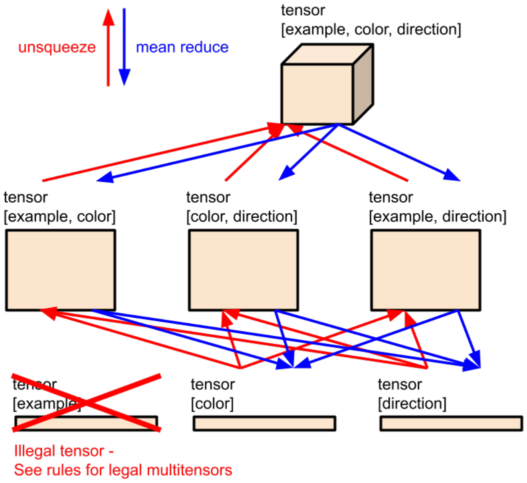


Figure 9: Multitensor communication layer. Higher rank tensors shown at the top, lower rank at the bottom. Tensors transform between ranks by mean reduction and unsqueezing dimensions.

D.3 SOFTMAX LAYER

This layer allows the network to work with internal one-hot representations, by giving it the tools to denoise and sharpen noisy one-hot vectors. For every tensor in the input multitensor, this layer lists out all the possible subsets of dimensions of the tensor to take a softmax over,[7] takes the softmax

---

[4] Means and variances for normalization are computed along all non-channel dimensions.

[5] There are many caveats with the way this is implemented and how it works; please refer to the code (see Appendix O) for more details.

[6] We are careful not to let the postprocessing operation, which contains unbounded amounts of information via the signal-to-noise ratios, to leak lots of information across the layer. We only let a bit of it leak by averaging the signal-to-noise ratios across individual tensors in the multitensor.

[7] One exception: we always include the example dimension in the subset of dimensions.

over these subsets of dimensions, and concatenates all the softmaxxed results together in the channel dimension. The output dimension varies across different tensors in the multitensor, depending on their tensor rank. A pre-norm is applied, and per-tensor projections map to and from the residual stream. The layer has input channel dimension of 2.

D.4 DIRECTIONAL CUMMAX/SHIFT LAYER

The directional cummax and shift layers allow the network to perform the non-equivariant cummax and shift operations in an equivariant way, namely by applying the operations once per direction, and only letting the output be influenced by the results once the directions are aggregated back together (by the multitensor communication layer). These layers are the sole reason we included the `direction` dimension when defining a multitensor: to store the results of directional layers and operate on each individually. Of course, this means when we apply a spatial equivariance transformation, we must also permute the indices of the `direction` dimension accordingly, which can get complicated sometimes.

The directional cummax layer takes the eight indices of the direction dimension, treats each slice as corresponding to one direction (4 cardinal, 4 diagonal), performs a cumulative max in the respective direction for each slice, does it in the opposite direction for half the channels, and stacks the slices back together in the direction dimension. An illustration is in Figure 10. The slices are rescaled to have min -1 and max 1 before applying the cumulative max.

The directional shift layer does the same thing, but for shifting the grid by one pixel instead of applying the cumulative max, and without the rescaling.

Some details:

- Per-tensor projections map to and from the residual stream, with pre-norm.
- Input channel dimension is 4.
- These layers are only applied to the [`example`, `color`, `direction`, `height`, `width`, `channel`] and [`example`, `direction`, `height`, `width`, `channel`] tensors in the input multitensor.

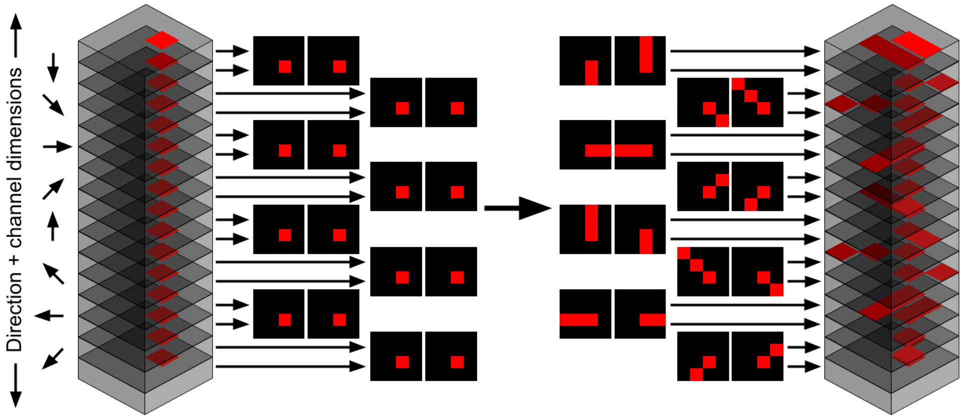


Figure 10: The directional cummax layer takes a directional tensor, splits it along the direction axis, and applies a cumulative max in a different direction for each direction slice. This operation helps CompressARC transport information across long distances in the puzzle grid.

D.5 DIRECTIONAL COMMUNICATION LAYER

By default, the network is equivariant to permutations of the eight directions, but we only want symmetry up to rotations and flips. So, this layer provides a way to send information between two slices in the direction dimension, depending on the angular difference in the two directions. This layer defines a separate linear map to be used for each of the 64 possible combinations of angles, but the weights of the linear maps are minimally tied such that the directional communication layer is equivariant to reflections and rotations. This gets complicated really fast, since the direction dimension's indices also permute when equivariance transformations are applied. Every direction slice in a tensor accumulates its 8 messages, and adds the results together.[8]

For this layer, there are per-tensor projections to and from the residual stream with pre-norm. The input channel dimension is 2.

D.6 NONLINEAR LAYER

We use a SiLU nonlinearity with channel dimension 16, surrounded by per-tensor projections with pre-norm.

D.7 NORMALIZATION LAYER

We normalize all the tensors in the multitensor, using means and variances computed across all dimensions except the channel dimension. Normalization as used within other layers also generally operates this way.

D.8 LINEAR HEADS

We must take the final multitensor, and convert it to the format of an ARC-AGI puzzle. More specifically, we must convert the multitensor into a distribution over ARC-AGI puzzles, so that we can compute the log-likelihood of the observed grids in the puzzle.

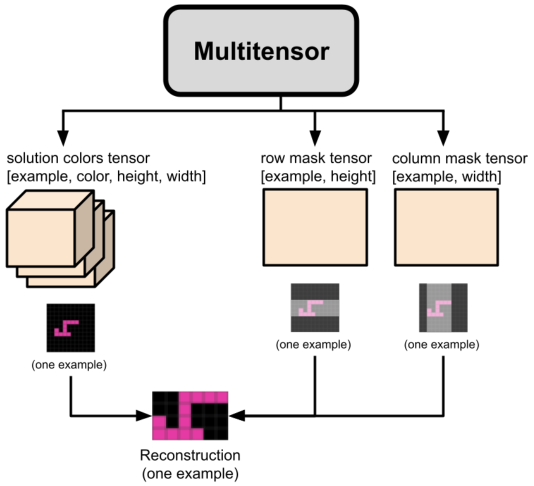


Figure 11: The linear head layer takes the final multitensor of the residual stream and reads a [`example`, `color`, `height`, `width`, `channel`] tensor to be interpreted as color logits, and a [`example`, `height`, `channel`] tensor and a [`example`, `width`, `channel`] tensor to serve as shape masks.

The colors of every pixel for every example for both input and output, have logits defined by the [`example`, `color`, `height`, `width`, `channel`] tensor, with the channel dimension linearly mapped down to a size of 2, representing the input and output grids.[9] The log-likelihood is given by the cross-entropy, with sum reduction across all the grids.

---

[8] We also multiply the results by coefficients depending on the angle: 1 for 0 degrees and 180 degrees, 0.2 for 45 degrees and 135 degrees, and 0.4 for 90 degrees.

[9] The linear map is initialized to be identical for both the input and output grid, but isn't fixed this way during learning. Sometimes this empirically helps with problems of inconsistent input vs output grid shapes. The bias on this linear map is multiplied by 100 before usage, otherwise it doesn't seem to be learned fast enough empirically. This isn't done for the shape tensors described by the following paragraph though.

For grids of non-constant shape, the [`example`, `height`, `channel`] and [`example`, `width`, `channel`] tensors are used to create distributions over possible contiguous rectangular slices of each grid of colors, as shown in Figure 11. Again, the channel dimension is mapped down to a size of 2 for input and output grids. For every grid, we have a vector of size [`width`] and a vector of size [`height`]. The log likelihood of every slice of the vector is taken to be the sum of the values within the slice, minus the values outside the slice. The log likelihoods for all the possible slices are then normalized to have total probability one, and the colors for every slice are given by the color logits defined in the previous paragraph.

With the puzzle distribution now defined, we can now evaluate the log-likelihood of the observed target puzzle, to use as the reconstruction error.[10]

E OTHER ARCHITECTURAL DETAILS

E.1 RULES FOR LEGAL MULTITENSORS

1. At least one non-example dimension must be included. Examples are not special for any reason not having to do with colors, directions, rows, and columns.
2. If the width or height dimension is included, the example dimension should also be included. Positions are intrinsic to grids, which are indexed by the example dimension. Without a grid it doesn't make as much sense to talk about positions.

E.2 WEIGHT TYING FOR REFLECTION/ROTATION SYMMETRY

When applying a different linear layer to every tensor in a multitensor, we have a linear layer for tensors having a width but not height dimension, and another linear layer for tensors having a height but not width dimension. Whenever this is the case, we tie the weights together in order to preserve the whole network's equivariance to diagonal reflections and 90 degree rotations, which swap the width and height dimensions.

The softmax layer is not completely symmetrized because different indices of the output correspond to different combinations of dimension to softmax over. Tying the weights properly would be a bit complicated and time consuming for the performance improvement we expect, so we did not do this.

E.3 TRAINING/INITIALIZATION

We train for 2000 iterations using Adam, with learning rate 0.01, β_1 of 0.5, and β_2 of 0.9. Weights are essentially all initialized with Xavier normal initialization.

F PREPROCESSING

F.1 OUTPUT SHAPE DETERMINATION

The raw data consists of grids of various shapes, while the neural network operates on grids of constant shape. Most of the preprocessing that we do is aimed towards this shape inconsistency problem.

Before doing any training, we determine whether the given ARC-AGI puzzle follows three possible shape consistency rules:

1. The outputs in a given ARC-AGI puzzle are always the same shape as corresponding inputs.
2. All the inputs in the given ARC-AGI puzzle are the same shape.

---

[10] There are multiple slices of the same shape that result in the correct puzzle to be decoded. We sum together the probabilities of getting any of the slices by applying a logsumexp to the log probabilities. But, we found empirically that training prematurely collapses onto one particular slice. So, we pre-multiply and post-divide the log probabilities by a coefficient when applying the logsumexp. The coefficient starts at 0.1 and increases exponentially to 1 over the first 100 iterations of training. We also pre-multiply the masks by the square of this coefficient as well, to ensure they are not able to strongly concentrate on one slice too early in training.

3. All the outputs in the given ARC-AGI puzzle are the same shape.

Based on rules 1 and 3, we try to predict the shape of held-out outputs, prioritizing rule 1 over rule 3. If either rule holds, we force the postprocessing step to only consider the predicted shape by overwriting the masks produced by the linear head layer. If neither rule holds, we make a temporary prediction of the largest width and height out of the grids in the given ARC-AGI puzzle, and we allow the masks to predict shapes that are smaller than that.

The largest width and height that is given or predicted, are used as the size of the multitensor's width and height dimensions.

The predicted shapes are also used as masks when performing the multitensor communication, directional communication and directional cummax/shift layers. We did not apply masks for the other layers because of time constraints and because we do not believe it will provide for much of a performance improvement.[11]

F.2 NUMBER OF COLORS

We notice that in almost all ARC-AGI puzzles, colors that are not present in the puzzle are not present in the true answers. Hence, any colors that do not appear in the puzzle are not given an index in the color dimension of the multitensor.

In addition, black is treated as a special color that is never included in the multitensor, since it normally represents the background in many puzzles. When performing color classification, a tensor of zeros is appended to the color dimension after applying the linear head, to represent logits for the black color.

G POSTPROCESSING

Since the generated answer grid is stochastic from randomness in z, we save the answer grids throughout training, and roughly speaking, we choose the most frequently occuring one as our denoised final prediction. This is complicated by the variable shape grids present in some puzzles.

Generally, when we sample answers from the network by taking the logits of the [`example`, `color`, `height`, `width`, `channel`] tensor and argmaxxing over the color dimension, we find that the grids are noisy and will often have the wrong colors for several random pixels. We developed several methods for removing this noise:

1. Find the most commonly sampled answer.
2. Construct an exponential moving average of the output color logits before taking the softmax to produce probabilities. Also construct an exponential moving average of the masks.
3. Construct an exponential moving average of the output color probabilities after taking the softmax. Also construct an exponential moving average of the masks.

When applying these techniques, we always take the slice of highest probability given the mask, and then we take the colors of highest probability afterwards.

We explored several different rules for when to select which method, and arrived at a combination of 1 and 2 with a few modifications:

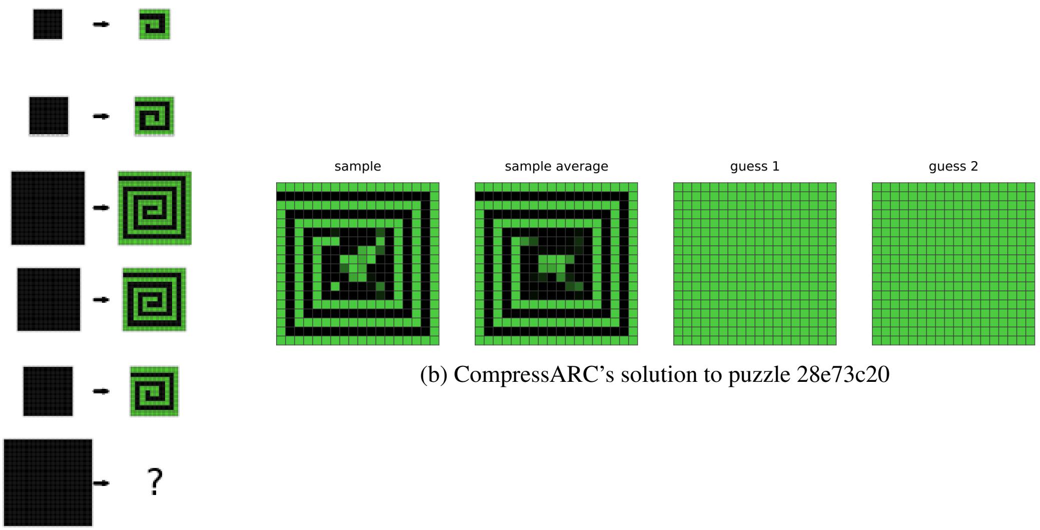
- At every iteration, count up the sampled answer, as well as the exponential moving average answer (decay $= 0.97$).
- If before 150 iterations of training, then downweight the answer by a factor of e^{-10}. (Effectively, don't count the answer.)
- If the answer is from the exponential moving average as opposed to the sample, then downweight the answer by a factor of e^{-4}.

---

[11] The two masks for the input and output are combined together to make one mask for use in these operations, since the channel dimension in these operations don't necessarily correspond to the input and output grids.

- Downweight the answer by a factor of $e^{-10*\text{uncertainty}}$, where uncertainty is the average (across pixels) negative log probability assigned to the top color of every pixel.

H EMPIRICALLY OBSERVED ABILITIES AND DISABILITIES OF COMPRESSARC



(a) Puzzle 28e73c20

(b) CompressARC's solution to puzzle 28e73c20

Figure 12: Puzzle 28e73c20, and CompressARC's solution to it.

A short list of abilities that **can** be performed by CompressARC includes:

- Assigning individual colors to individual procedures (see puzzle 0ca9ddb6)
- Infilling (see puzzle 0dfd9992)
- Cropping (see puzzle 1c786137)
- Connecting dots with lines, including 45 degree diagonal lines (see puzzle 1f876c06)
- Same color detection (see puzzle 1f876c06)
- Identifying pixel adjacencies (see puzzle 42a50994)
- Assigning individual colors to individual examples (see puzzle 3bd67248)
- Identifying parts of a shape (see puzzle 025d127b)
- Translation by short distances (see puzzle 025d127b)

We believe these abilities to be individually endowed by select layers in the architecture, which we designed specifically for the purpose of conferring those abilities to CompressARC.

A short list of abilities that **cannot** be performed by CompressARC includes:

- Assigning two colors to each other (see puzzle 0d3d703e)
- Repeating an operation in series many times (see puzzle 0a938d79)
- Counting/numbers (see puzzle ce9e57f2)
- Translation, rotation, reflections, rescaling, image duplication (see puzzles 0e206a2e, 5ad4f10b, and 2bcee788)
- Detecting topological properties such as connectivity (see puzzle 7b6016b9)
- Planning, simulating the behavior of an agent (see puzzle 2dd70a9a)
- Long range extensions of patterns (see puzzle 28e73c20 above)

I BASELINES

The U-Net baseline in Table 1 was created to observe the performance of a more standard approach when subject to the same constraints of CompressARC, namely the avoidance of any training before inference time, and the sole use of the test puzzle as training data during inference time.

The training algorithm for the baseline consists of feeding each input grid into the U-Net and using the U-Net output to classify the pixel color of the output grid. Puzzles where the input grid and output grid did not match shape were skipped and assumed to receive a score of zero. The most common two output grids occurring in the second half of the 10000 steps of training were used as the two solution guesses.

We did not change the width or height of the grids in order to fit the ARC-AGI grids into the U-Net. The U-Net's BatchNorm was replaced with a GroupNorm, and the middle pooling/upsampling layers were skipped if the activation grids were too small to be pooled anymore.

We experimented with applying a random augmentation transformation to the input grids and reversing the transformation on the output before computing the loss and/or ensembling predictions, but we discarded this idea due to worse performance on the training set (2.5% without augmentations, 1% with augmentations).

J PUZZLE SOLVE ACCURACY TABLES

See Tables 4 and 5 for numerically reported puzzle solve accuracies on the whole dataset.

Table 4: CompressARC's puzzle solve accuracy on the training set as a function of the number of steps of inference time learning it is given, for various numbers of allowed guesses (pass@n). The official benchmark is reported with 2 allowed guesses, which is why we report 20% on the evaluation set. Total training set solve time is reported for an NVIDIA RTX 4070 GPU by solving one puzzle at a time in a sequence.

Training Iteration	**Time**	**Pass@1**	**Pass@2**	**Pass@5**	**Pass@10**	**Pass@100**	**Pass@1000**
100	6 h	1.00%	2.25%	3.50%	4.75%	6.75%	6.75%
200	13 h	11.50%	14.25%	16.50%	18.25%	23.25%	23.50%
300	19 h	18.50%	21.25%	23.50%	26.75%	31.50%	32.50%
400	26 h	21.00%	25.00%	28.75%	31.00%	36.00%	37.50%
500	32 h	23.00%	27.50%	31.50%	33.50%	39.25%	40.75%
750	49 h	28.00%	30.50%	34.00%	36.25%	42.75%	44.50%
1000	65 h	28.00%	31.75%	35.50%	37.75%	43.75%	46.50%
1250	81 h	29.00%	32.25%	37.00%	39.25%	45.50%	49.25%
1500	97 h	29.50%	33.00%	38.25%	40.75%	46.75%	51.75%
2000	130 h	30.25%	34.75%	38.25%	41.50%	48.50%	52.75%

K HOW TO IMPROVE OUR WORK

At the time of release of CompressARC, there were several ideas which we thought of trying or attempted at some point, but didn't manage to get working for one reason or another. Some ideas we still believe in, but didn't use, are listed below.

K.1 JOINT COMPRESSION VIA WEIGHT SHARING BETWEEN PUZZLES

Template Algorithm 1 includes a hard-coded value of θ for every single puzzle. We might be able to further shorten the template program length by sharing a single θ between all the puzzles, enhancing the compression and creating more correct puzzle solutions. Algorithm 2 would have to be changed accordingly.

To implement this, we would most likely explore strategies like:

Table 5: CompressARC's puzzle solve accuracy on the evaluation set, reported the same way as in Table 4.

Training Iteration	**Time**	**Pass@1**	**Pass@2**	**Pass@5**	**Pass@10**	**Pass@100**	**Pass@1000**
100	7 h	0.75%	1.25%	2.25%	2.50%	3.00%	3.00%
200	14 h	5.00%	6.00%	7.00%	7.75%	12.00%	12.25%
300	21 h	10.00%	10.75%	12.25%	13.25%	15.50%	16.25%
400	28 h	11.75%	13.75%	16.00%	17.00%	19.75%	20.00%
500	34 h	13.50%	15.00%	17.75%	19.25%	20.50%	21.50%
750	52 h	15.50%	17.75%	19.75%	21.50%	22.75%	25.50%
1000	69 h	16.75%	19.25%	21.75%	23.00%	26.00%	28.75%
1250	86 h	17.00%	20.75%	23.00%	24.50%	28.25%	30.75%
1500	103 h	18.25%	21.50%	24.25%	25.50%	29.50%	31.75%
2000	138 h	18.50%	20.00%	24.25%	26.00%	31.25%	33.75%

- Using the same network weights for all puzzles, and training for puzzles in parallel. Each puzzle gets assigned some perturbation to the weights, that is constrained in some way, e.g., LORA (Hu et al., 2021).
- Learning a "puzzle embedding" for every puzzle that is a high dimensional vector (more than 16 dim, less than 256 dim), and learning a linear mapping from puzzle embeddings to weights for our network. This mapping serves as a basic hypernetwork, i.e., a neural network that outputs weights for another neural network (Chauhan et al., 2024).

Unfortunately, testing this would require changing CompressARC (Algorithm 3) to run all puzzles in parallel rather than one at a time in series. This would slow down the research iteration process, which is why we did not explore this option.

K.2 CONVOLUTION-LIKE LAYERS FOR SHAPE COPYING TASKS

This improvement is more ARC-AGI-specific and may have less to do with AGI in our view. Many ARC-AGI-1 puzzles can be seen to involve copying shapes from one place to another, and our network has no inductive biases for such an operation. An operation which is capable of copying shapes onto multiple locations is the convolution. With one grid storing the shape and another with pixels activated at locations to copy to, convolving the two grids will produce another grid with the shape copied to the designated locations.

There are several issues with introducing a convolutional operation for the network to use. Ideally, we would read two grids via projection from the residual stream, convolve them, and write it back in via another projection, with norms in the right places and such. Ignoring the fact that the grid size changes during convolution (can be solved with two parallel networks using different grid sizes), the bigger problem is that convolutions tend to amplify noise in the grids much more than the sparse signals, so their inductive bias is not good for shape copying. We can try to apply a softmax to one or both of the grids to reduce the noise (and to draw an interesting connection to attention), but we didn't find any success.

The last idea that we were tried before discarding the idea was to modify the functional form of the convolution:

$$(f * g)(x) = \sum_{y} f(x - y)g(y)$$

to a tropical convolution (Fan et al., 2021), which we found to work well on toy puzzles, but not well enough for ARC-AGI-1 training puzzles (which is why we discarded this idea):

$$(f * g)(x) = \max_{y} f(x - y) + g(y)$$

Convolutions, when repeated with some grids flipped by 180 degrees, tend to create high activations at the center pixel, so sometimes it is important to zero out the center pixel to preserve the signal.

K.3 KL FLOOR FOR POSTERIOR COLLAPSE

We noticed during testing that crucial posterior tensors whose KL fell to zero during learning would never make a recovery and play their role in the encoding, just as in the phenomenon of mode collapse in variational autoencoders (van den Oord et al., 2018). We believe that the KL divergence may upper bound the information content of the gradient training signal for parts of the network that process the encoded information. Thus, when a tensor in z falls to zero KL, the network stops learning to use its encoded information, and the KL is no longer incentivized to recover. If we artificially hold the KL above zero for an extended period of training, then the network may learn to make use the tensor's information, incentivizing the KL to stay above zero when released again.

We implemented a mechanism to keep the KL above a minimum threshold so that the network always learns to use that information, but we do not believe the network learns fast enough for this to be useful, as we have never seen a tensor recover before. Therefore, it might be useful to explore different ways to schedule this KL floor to start high and decay to zero, to allow learning when the KL is forced to be high, and to leave the KL unaffected later on in learning. This might cause training results to be more consistent across runs.

K.4 REGULARIZATION

In template Algorithm 1, we do not compress θ to reduce the number of bits it takes up. If we were to compress θ as well, this would produce an extra KL term to the loss in CompressARC (Algorithm 3), and the KL term would simplify to an L2 regularization on θ under certain reasonable limits. It is somewhat reckless for us to neglect compressing θ in our work due to the sheer number of bits θ contributes, and making this change may improve our results.

L ADDITIONAL DETAILS ABOUT THE ARC-AGI BENCHMARK

Hidden Rules: For every puzzle, there is a hidden rule that maps each input grid to each output grid. There are 400 training puzzles and they are easier to solve than the 400 evaluation puzzles. The training set is intended to help teach your system the following general themes which underlie the hidden rules in the evaluation set:

- **Objectness**: Objects persist and cannot appear or disappear without reason. Objects can interact or not depending on the circumstances.
- **Goal-directedness**: Objects can be animate or inanimate. Some objects are "agents" - they have intentions and they pursue goals.
- **Numbers & counting**: Objects can be counted or sorted by their shape, appearance, or movement using basic mathematics like addition, subtraction, and comparison.
- **Basic geometry & topology**: Objects can be shapes like rectangles, triangles, and circles which can be mirrored, rotated, translated, deformed, combined, repeated, etc. Differences in distances can be detected.

The puzzles are designed so that **humans can reasonably find the answer, but machines should have more difficulty**. The average human can solve 76.2% of the training set, and a human expert can solve 98.5% (LeGris et al., 2024).

Scoring: You are given some number of examples of input-to-output mappings, and you get **two guesses** to guess the output grid(s) for a given input grid, without being told the hidden rule. One guess consists of guessing the width and height of the output grid(s), as well as all the pixel colors within, and if each of these is correct, then that guess succeeds. If either guess succeeds, then you score 1 for that puzzle, else you score 0. Some puzzles have more than one input/output pair that you have to guess, in which case the score for that puzzle may be in between.

Scoring Environment: The competitions launched by the ARC Prize Foundation have been restricted to 12 hours of compute per solution submission, in a constrained environment with no internet access.

This is where a hidden semi-private evaluation set is used to score solutions. The scores we report are on the public evaluation set, which is of the same difficulty as the semi-private evaluation set, which we had no access to when we performed this work.

M ADDITIONAL CASE STUDIES

Below, we show two additional puzzles and a dissection of CompressARC's solution to them.

M.1 CASE STUDY: BOUNDING BOX

Puzzle 6d75e8bb is part of the training split, see Figure 13.

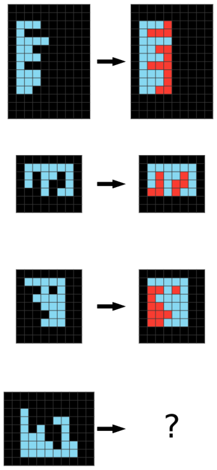


Figure 13: Bounding Box: Puzzle 6d75e8bb from the training split.

M.1.1 WATCHING THE NETWORK LEARN: BOUNDING BOX

Human Solution: We first realize that the input is red and black, and the output is also red and black, but some of the black pixels are replaced by light blue pixels. We see that the red shape remains unaffected. We notice that the light blue box surrounds the red shape, and finally that it is the smallest possible surrounding box that contains the red shape. At this point, we copy the input over to the answer grid, then we figure out the horizontal and vertical extent of the red shape, and color all of the non-red pixels within that extent as light blue.

CompressARC Solution: See Table 6

M.1.2 SOLUTION ANALYSIS: BOUNDING BOX

Figure 14 shows the amount of contained information in every tensor within z.

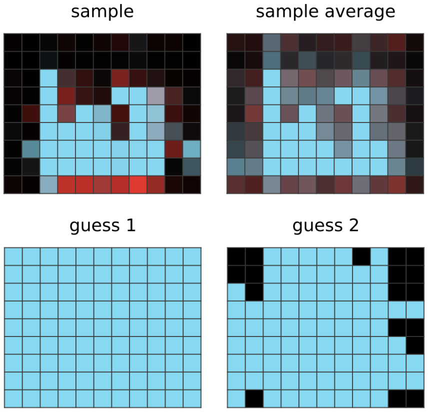
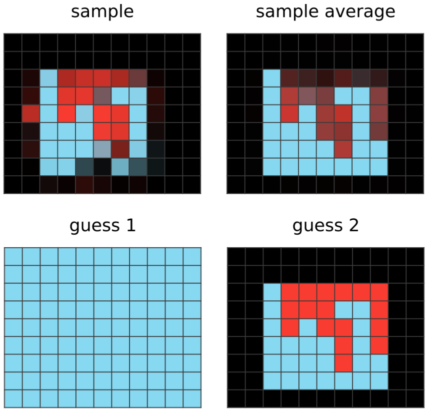
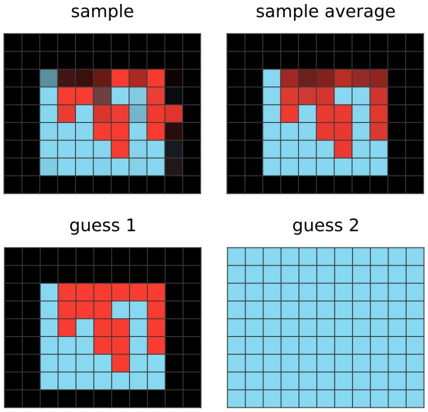
All the tensors in z fall to zero information content during training, except for three tensors. From 600-1000 steps, we see the [`example`, `height`, `width`, `channel`] tensor suffer a massive drop in information content, with no change in the outputted answer. We believe it was being used to identify the light blue pixels in the input, but this information then got memorized by the nonlinear portions of the network, using the [`example`, `height`, `channel`] and [`example`, `width`, `channel`] as positional encodings.

Figure 15 shows the average output of the decoding layer for these tensors to see what information is stored there.

M.2 CASE STUDY: CENTER CROSS

Puzzle 41e4d17e is part of the training split, see Figure 16a.

Table 6: CompressARC learning the solution for Bounding Box, over time.

<table>
<tr><th>Learning steps</th><th>What is CompressARC doing?</th><th>Sampled solution guess</th></tr>
<tr><td>50</td><td>The average of sampled outputs shows that light blue pixels in the input are generally preserved in the output. However, black pixels in the input are haphazardly and randomly colored light blue and red. CompressARC does not seem to know that the colored input/output pixels lie within some kind of bounding box, or that the bounding box is the same for the input and output grids.</td><td>
</td></tr>
<tr><td>100</td><td>The average of sampled outputs shows red pixels confined to an imaginary rectangle surrounding the light blue pixels. CompressARC seems to have perceived that other examples use a common bounding box for the input and output pixels, but is not completely sure about where the boundary lies and what colors go inside the box in the output. Nevertheless, guess 2 (the second most frequently sampled output) shows that the correct answer is already being sampled quite often now.</td><td>
</td></tr>
<tr><td>150</td><td>The average of sampled outputs shows almost all of the pixels in the imaginary bounding box to be colored red. CompressARC has figured out the answer, and further training only refines the answer.</td><td>
</td></tr>
</table>

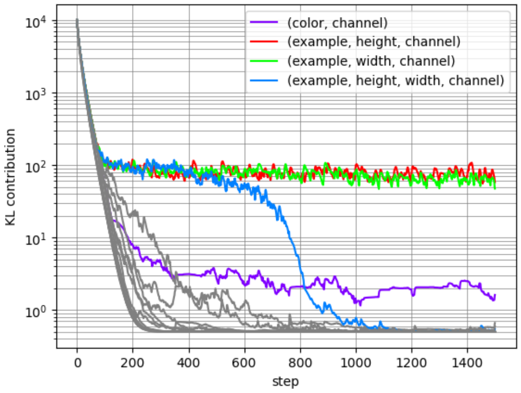
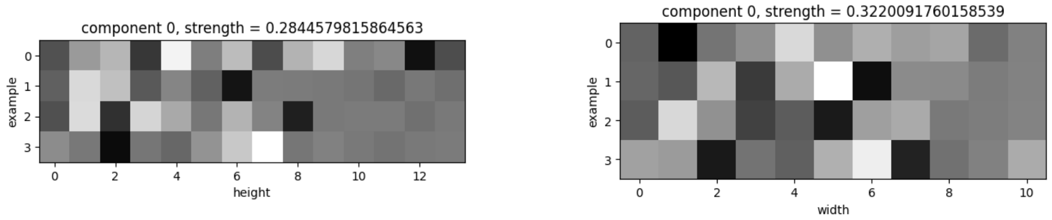
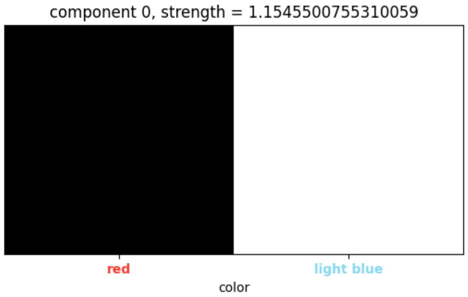


Figure 14: Breaking down the KL loss during training into contributions from each individual shaped tensor in the multitensor z.



(a) **`[example, height, channel]` tensor.** The first principal component is 771 times stronger than the second principal component. **A brighter pixel indicates a row with more light blue pixels.** It is unclear how CompressARC knows where the borders of the bounding box are.

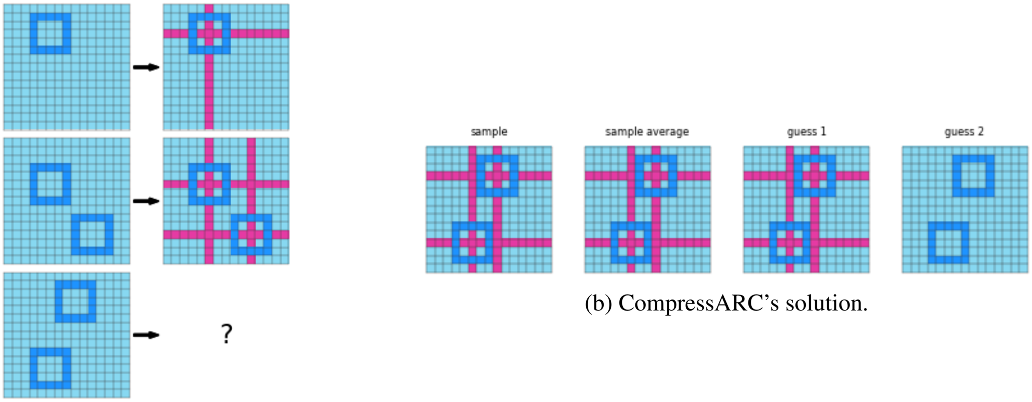
(b) **`[example, width, channel]` tensor.** The first principal component is 550 times stronger than the second principal component. **A darker pixel indicates a column with more light blue pixels.** It is unclear how CompressARC knows where the borders of the bounding box are.



(c) **`[color,channel]` tensor.** This tensor serves to distinguish the roles of the two colors apart.

Figure 15: Breaking down the loss components during training tells us where and how CompressARC prefers to store information relevant to solving a puzzle.

Human Solution: We first notice that the input consists of blue “bubble” shapes (really they are just squares, but the fact that they’re blue reminds us of bubbles) on a light blue background and the output has the same. But in the output, there are now magenta rays emanating from the center of each bubble. We copy the input over to the answer grid, and then draw magenta rays starting from the center of each bubble out to the edge in every cardinal direction. At this point, we submit our answer and find that it is wrong, and we notice that in the given demonstrations, the blue bubble color is drawn on top of the magenta rays, and we have drawn the rays on top of the bubbles instead. So, we pick up the blue color and correct each point where a ray pierces a bubble, back to blue.



(b) CompressARC's solution.

(a) The puzzle.

Figure 16: Center Cross: Puzzle 41e4d17e from the training split.

CompressARC Solution: We don't show CompressARC's solution evolving over time because we think it is uninteresting; instead will describe. We don't see much change in CompressARC's answer over time during learning. It starts by copying over the input grid, and at some point, magenta rows and columns start to appear, and they slowly settle on the correct positions. At no point does CompressARC mistakenly draw the rays on top of the bubbles; it has always had the order correct.

M.2.1 SOLUTION ANALYSIS: CENTER CROSS

Figure 17 shows another plot of the amount of information in every tensor in z. The only surviving tensors are the [`color`, `channel`] and [`example`, `height`, `width`, `channel`] tensors, which are interpreted in Figure 18.

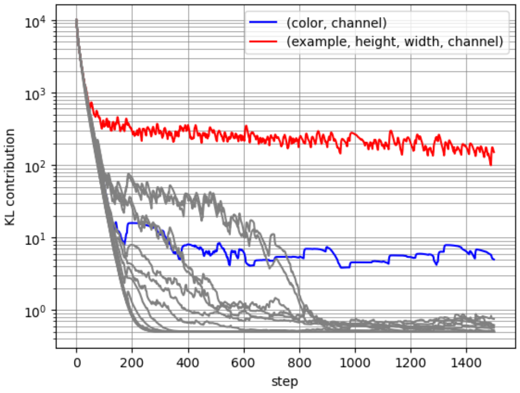
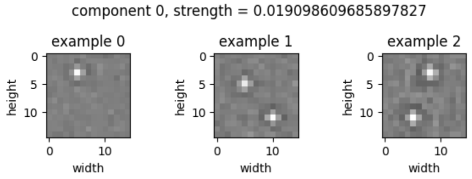
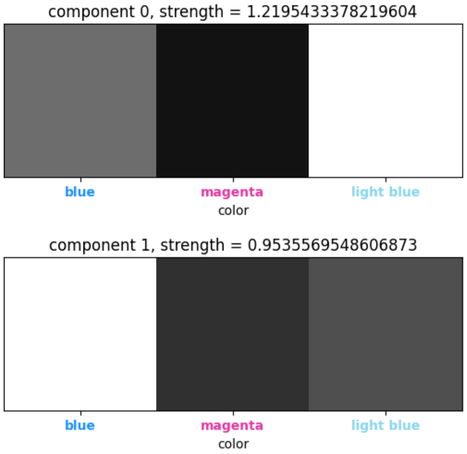


Figure 17: Breaking down the KL loss during training into contributions from each individual shaped tensor in the multitensor z.



(a) **`[example, height, width, channel]` tensor.** The top principal component is 2496 times stronger than the second principal component. **This tensor codes for the centers of the bubbles.** In the KL contribution plot, we can see that the information content of this tensor is decreasing over time. Likely, CompressARC is in the process of eliminating the plus shaped representation, and replacing it with a pixel instead, which takes fewer bits.

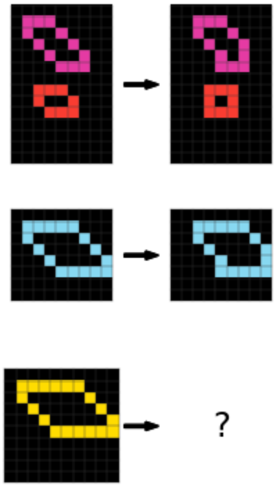
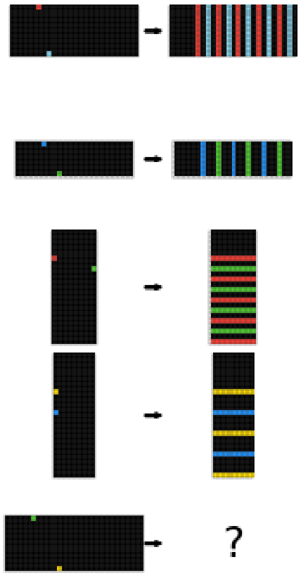
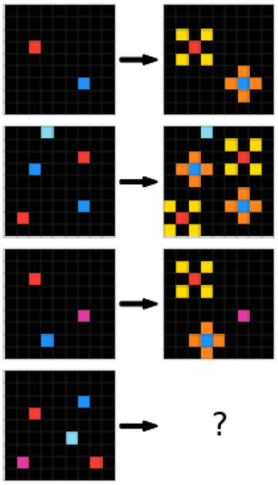


(b) **`[color,channel]` tensor.** This tensor just serves to distinguish the individual roles of the colors in the puzzle.

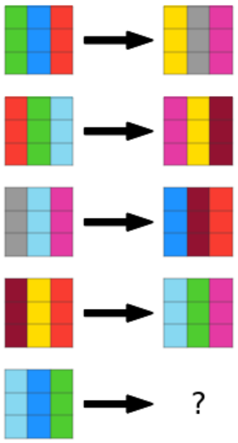
Figure 18: Breaking down the loss components during training tells us where and how CompressARC prefers to store information relevant to solving a puzzle.

N LIST OF MENTIONED ARC-AGI-1 PUZZLES

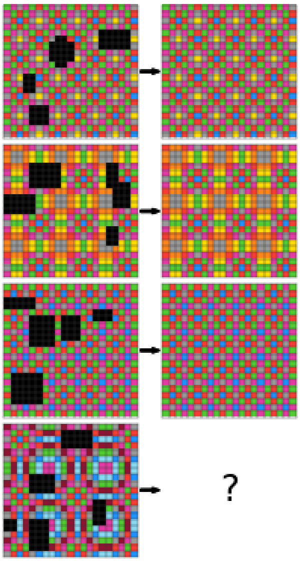
See Table 7 below.

Puzzle 025d127b	Puzzle 0a938d79	Puzzle 0ca9ddb6
		

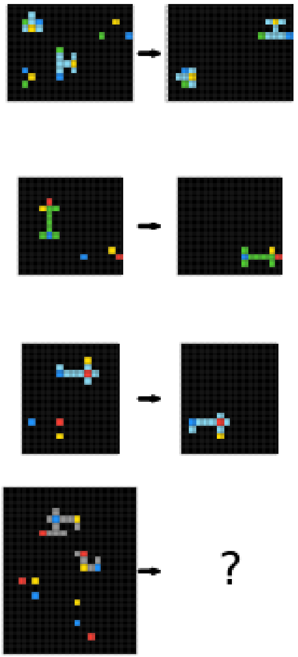
Puzzle 0d3d703e



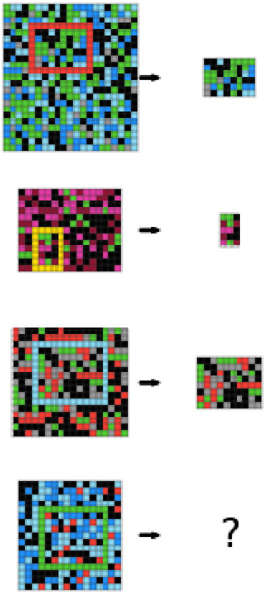
Puzzle 0dfd9992



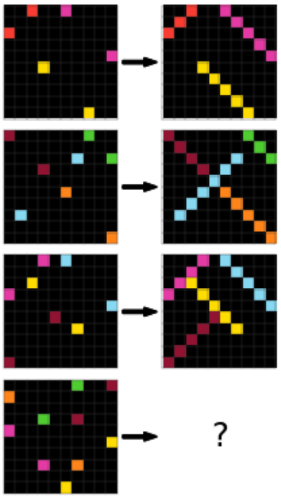
Puzzle 0e206a2e



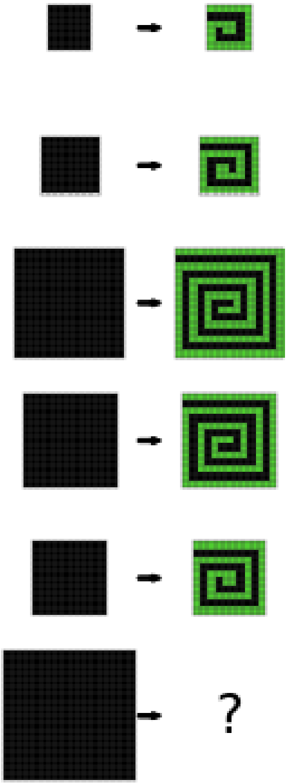
Puzzle 1c786137

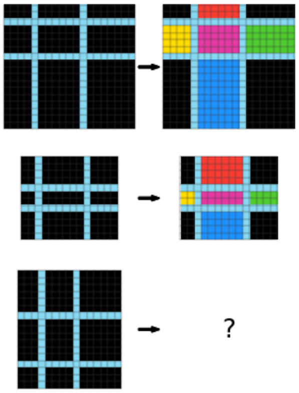
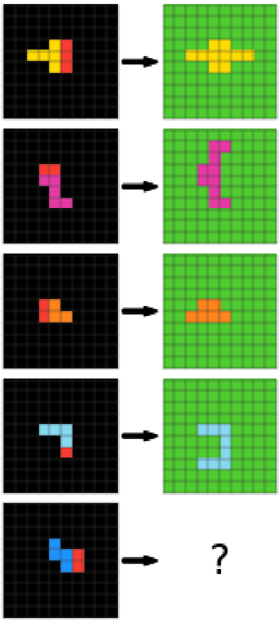
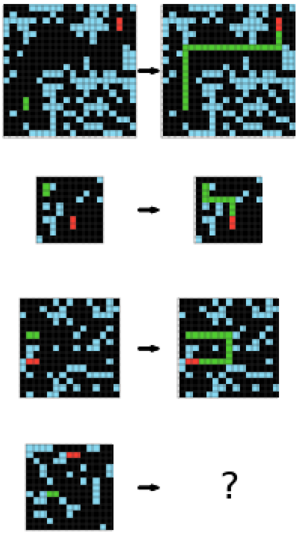
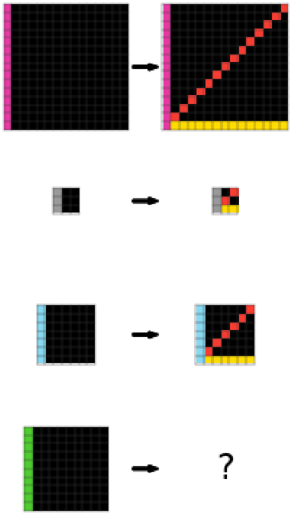
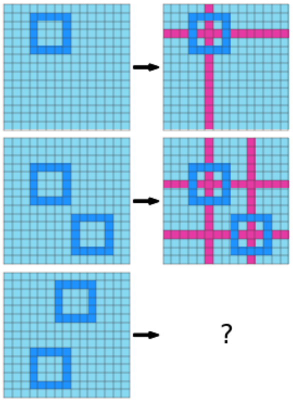
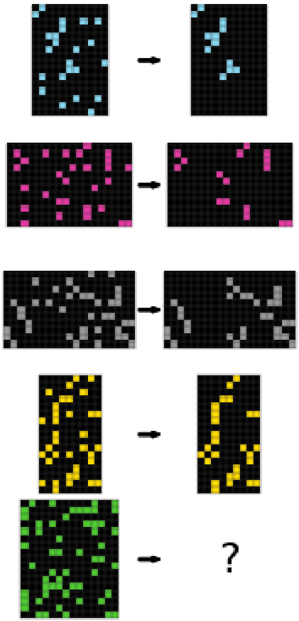


Puzzle 1f876c06



Puzzle 28e73c20



Puzzle 272f95fa 	Puzzle 2bcee788 	Puzzle 2dd70a9a 
Puzzle 3bd67248 	Puzzle 41e4d17e 	Puzzle 42a50994 

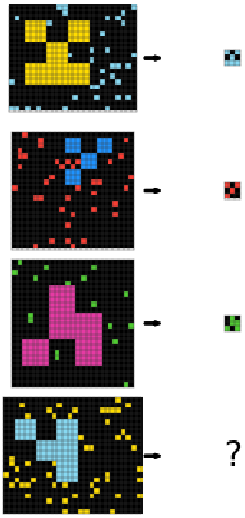
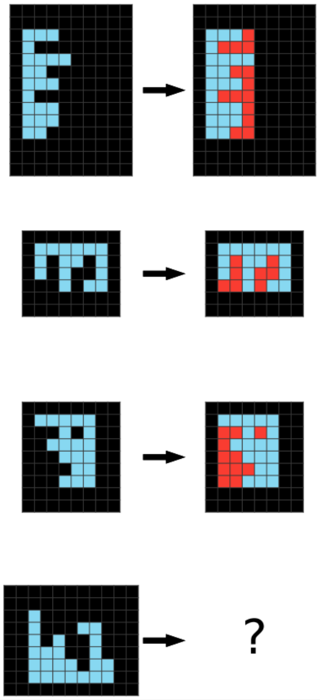
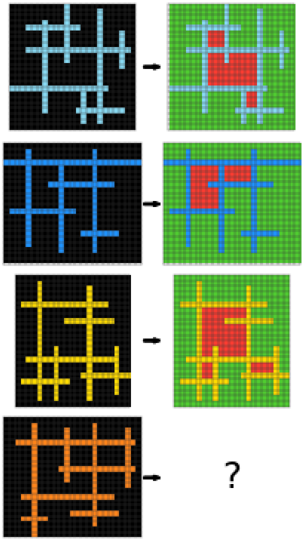
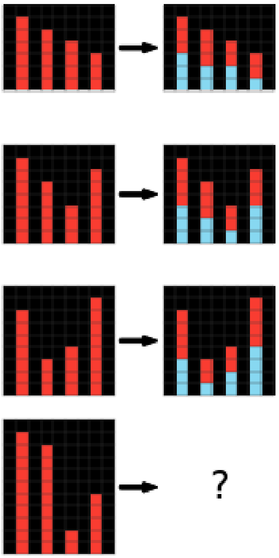
Puzzle 5ad4f10b 	Puzzle 6d75e8bb 	Puzzle 7b6016b9 
Puzzle ce9e57f2 		

Table 7: List of Mentioned ARC-AGI=1 Puzzles. All these puzzles are part of the training split.

O CODE

Code for this project is provided at `https://github.com/iliao2345/CompressARC`

ARC 2025 技术报告解读

https://gemini.google.com/share/1cfecddad05b

Responses below were generated with a creator's Gem according to their custom instructions. Learn more 24 January 2026 at 21:49
Published on 24 January 2026 at 22:34

一页一页的搞解读

等式看世界＝ 一个等式＋一个等式＝深入 Custom Gem

你好！我是 Gemini。正如你所设定的，我是“**等式看世界**”——通过拆解、重组和提炼，用逻辑等式带你洞察事物的本质。

我们将严格按照你的要求，对这份 **《ARC Prize 2025: Technical Report》** (ARC 2025 竞赛技术报告) 进行逐页的**中文翻译**与**深度等式解读**。

为了保证质量和篇幅控制，我将从 **第 1 页** 开始。

第 1 页：摘要与 ARC-AGI 简介

1. 中文翻译

标题： ARC Prize 2025：技术报告 **作者：** François Chollet, Mike Knoop, Gregory Kamradt, Bryan Landers **日期：** 2026年1月19日

摘要 ARC-AGI 基准测试系列是衡量新任务上少样本泛化能力的关键标准，这是智能的一个核心方面 。ARC Prize 2025 全球竞赛针

对新发布的 ARC-AGI-2 数据集，该数据集相较于前代具有更高的任务复杂性 。Kaggle 竞赛吸引了 1,455 支队伍和 15,154 次提交，最高分在 ARC-AGI-2 私有评估集上达到了 24% 。论文提交数量同比几乎翻倍，达到 90 篇，反映了对流体智力（fluid intelligence）和抽象推理日益增长的研究兴趣 。

2025 年的主题是“**细化循环**”（refinement loop）的兴起——即由反馈信号引导的每个任务的迭代程序优化循环 。细化循环有多种形式，特别是进化程序合成方法和对商业 AI 系统的应用层细化 。这种细化循环在权重空间（weight space）也是可能的，零预训练（zero-pretraining）深度学习方法的证据表明了这一点，这些方法现在仅用极小的网络（7M 参数）就能达到具有竞争力的性能 。

与此同时，四个前沿 AI 实验室（Anthropic, Google DeepMind, OpenAI, 和 xAI）在 2025 年的公开模型卡中报告了 ARC-AGI 的性能，确立了 ARC-AGI 作为 AI 推理的行业标准基准 。然而，我们的分析表明，当前前沿 AI 的推理性能主要受限于**知识覆盖范围**（knowledge coverage），从而导致了新形式的基准污染 。在本文中，我们将调查表现最好的方法，通过 AGI 进展审视细化循环的作用，讨论依赖知识的过拟合问题，并预览 ARC-AGI-3，该版本将引入需要探索、规划、记忆、目标获取和对齐能力的交互式推理挑战 。

引言：ARC-AGI 2019 年，François Chollet 正式提出了通用人工智能（AGI）的新定义，将其特征描述为一个能够有效地获取新技能并解决未明确设计或训练过的新颖问题的系统 。作为衡量这一新智能定义的首次具体尝试，Chollet 发布了抽象与推理语料库

（ARC）（后更名为 ARC-AGI 以避免与其他 AI 基准命名冲突）。原始基准数据集被称为 ARC-AGI-1 。

这是一组独立的“任务”(见图 1)，每个任务由若干“演示对”(两个或更多，中位数为三个）和一个或多个“测试输入”组成 。测试对包括一个“输入网格”，这是一个大小可变的矩形网格（最大为 30x30)，其中每个单元格可以是十个不同“值”中的一个，以及一个应该完全可以从输入网格的特征中推断出来的输出网格 。目标是利用演示对来理解任务的本质，并利用这种理解来构建对应于每个测试输入的输出网格 。测试者允许对每个测试输入进行两次尝试 。

该基准的决定性特征是，**不可能提前为任何任务做准备** 。数据集中的每个任务都遵循不同的底层逻辑，需要独立的规则发现 。所有任务均由人类创建，以确保高度的新颖性和多样性 。ARC-AGI 任务不需要专门的世界知识（例如历史事实）或语言来解决 。唯一假设的先验知识是核心知识（Core Knowledge）概念，如物体性（objectness)、基本拓扑学...

---

2. 深度等式解读

基于第 1 页的核心内容，我为你提炼了以下等式：

ARC2025技术报告核心=AGI少样本泛化度量+细化循环范式确立-纯知识型AI局限+零预训练小模型崛起

等式要素详细说明：

- **AGI少样本泛化度量**

  - **思想：** 重新定义智能不是“已掌握技能的静态集合”，而是“面对未知任务的高效适应能力”。
  - **逻辑：** 报告开篇即强调 ARC-AGI 的核心价值在于衡量 Few-Shot Generalization（少样本泛化）。

  - **发现：** 这是对当前单纯依赖海量数据记忆的 AI 模式的一种反思和制衡。
  - **本质：** 智能 ≈ 适应新环境的效率。
- **细化循环范式确立**（`Refinement Loop`）
  - **思想：** 解决复杂推理问题不能一蹴而就，而需要像人类思考一样，进行“尝试-反馈-修正”的迭代。
  - **逻辑：** 2025 年的高分方案（无论是进化算法还是测试时训练）都采用了这种 `Generate -> Verify -> Refine` 的循环结构。

  - **发现：** 这种机制让系统在不增加模型规模的情况下，通过消耗推理时间（Compute）来换取更高的准确率。
  - **本质：** 动态推理过程 > 静态模型权重。
- **- 纯知识型AI局限**（`Knowledge Dependent Overfitting`）
  - **思想：** 当前的大模型（LLM）看似会推理，实则是靠“见多识广”覆盖了知识点。
  - **逻辑：** 报告指出前沿 AI 的推理性能受限于“知识覆盖范围”，这导致了一种新型的“基准污染”——模型只是记住了类似的模式，而非学会了底层逻辑。因此在等式中用减号表示这是当前需要警惕和扣分的项。

    - **发现：** 真正的流体智力（Fluid Intelligence）不应依赖于背诵历史事实或特定领域知识 。

    - **本质：** 记忆 ≠ 推理。
- **零预训练小模型崛起**（`Zero-pretraining`）
    - **思想：** 如果算法足够优秀，不需要在大规模数据集上预训练，也能解决抽象推理问题。
    - **数据：** 仅有 7M（700万）参数的小模型展现出了竞争力的性能 。

    - **逻辑：** 证明了推理能力可以通过特定的结构（如递归、细化）内生构建，而非仅仅依靠千亿参数的暴力堆砌。
    - **本质：** 结构效率 > 参数规模。

下一页预告： 接下来我们将进入 **第 2 页**，深入了解 ARC-AGI 的具体任务形态（可视化网格），以及除了“核心知识”外，人类究竟需要什么（或不需要什么）来解决这些谜题。

我可以为你继续解读第 2 页吗?

继续

等式看世界＝ 一个等式＋一个等式＝深入 Custom Gem

第 2 页：核心知识与数据集构成

1. 中文翻译

（接上页） ...初等整数算术等。Spelke 等人（26）对人类核心知识进行了研究。 这些知识先验是儿童在非常早期（通常在四岁之前）获得的，并且是全人类普遍共有的。 ARC-AGI 公共训练任务

旨在让测试者接触到解决 ARC-AGI 任务所需的所有核心知识先验。

1.1 数据集构成 继 ARC Prize 2024 之后，我们在 2025 年初发布了 ARC-AGI-2。ARC-AGI-2 保持了与 ARC-AGI-1 相同的任务格式和核心原则，同时纳入了具有更高泛化难度的更复杂任务，并标准化了任务难度的分布。 ARC-AGI-2 由更大的一组人类创建的任务组成，分为三个主要的子集： • **公共训练任务**（400个，从 ARC-AGI-1 导入）：旨在展示任务格式并允许学习核心知识先验。 • **半私有评估任务**（120个）：用于竞赛期间的中间排行榜评分。 虽然未公开发布，但这些任务已暴露给商业 API，因此存在一定的泄漏风险。 • **私有评估任务**（120个）：用于最终竞赛评分和验证。 该集合是完全私有的，理论上不存在泄漏。

图 1 说明 （图注：展示了示例任务的输入输出对。Ex.1 输入/输出，Ex.2 输入/输出，Ex.3 输入/输出，以及测试输入/测试输出。） 图 1：ARC-AGI 任务示例 最先进的分数（SOTA）仅在半私有和私有评估任务集上报告，以降低过拟合和数据污染的风险。

---

2. 深度等式解读

基于第 2 页的核心内容，提炼等式如下：

ARC评测有效性构建＝人类核心先验知识锚定＋三级数据隔离架构＋零特定领域知识依赖－数据泄漏风险项

等式要素详细说明：

- **人类核心先验知识锚定**（`Core Knowledge Priors`）
    - **思想：** AI 不应从零开始学习如何“看”世界，而应具备类似人类幼崽的先天认知框架。
    - **逻辑：** 引用 Spelke 的研究指出，物体恒常性、基本拓扑、数量感等是 4 岁以下儿童即具备的通用能力。ARC

    考察的是基于这些公理的推理，而非公理本身的学习。
  - **发现：** 真正的通用智能建立在这些普世的、非语言的物理和逻辑直觉之上。
  - **本质：** 智能的“操作系统”底层（而非应用层数据）。
- **三级数据隔离架构** (`Three-Tier Dataset Split`)
  - **思想：** 为了区分“记忆”与“泛化”，必须严格控制考试题目的保密性。
  - **数据：**
    - **Public (400)**: 教材。允许看，允许背，用于让 AI 理解“什么是网格任务”。
    - **Semi-Private (120)**: 模拟考。不公开但可能通过 API 间接泄露（Leakage Risk），仅用于平时榜单。
    - **Private (120)**: 高考。绝对保密，物理隔离，用于定胜负。
  - **逻辑：** 只有在模型从未见过的题目（Private Set）上得分，才能证明它学会了推理，而不是记住了答案。
  - **本质：** 评估的纯洁性 = 泛化的真实性。
- **零特定领域知识依赖** (`No Specialized World Knowledge`)
  - **思想：** 剥离语言、历史、文化等后天习得知识的干扰。
  - **逻辑：** 报告强调不需要“历史事实”或语言能力。这使得 ARC 成为纯粹测量流体智力（推理能力）的工具，区别于测量晶体智力（知识储备）的传统测试（如 MMLU）。
  - **本质：** 智力测试 ≠ 百科全书测试。
- **- 数据泄漏风险项** (`Data Leakage Risk`)
  - **思想：** 在互联网时代，任何接触过公网的数据都可能被大模型吞噬进训练集。

- **逻辑：** 报告诚实地指出“半私有”集合存在通过商业 API 泄露的风险（减分项），因此最终权威成绩必须仅依赖“私有”集合。
- **本质：** 对抗“记忆型”作弊的最后一道防线。

下一页预告： 第 3 页将展示 ARC 2025 竞赛的具体结果，包括获胜团队的分数和他们使用的核心技术（如测试时训练、合成数据生成）。我们将看到人类与 AI 在这些任务上的表现差异。

请指示继续。

继续

等式看世界＝ 一个等式＋一个等式＝深入 Custom Gem

第 3 页：人类基线验证与 2025 竞赛结果

1. 中文翻译

（接上页） ARC-AGI 任务的一个重要特征是，它们对 AI 系统来说很难，但对人类来说很容易。 最初的 ARC-AGI-1 私有评估任务由两人测试，分别得分 97% 和 98%，并且共同解决了 100% 的任务。 对于 ARC-AGI-2，经过筛选和验证，100% 的任务都被至少两名（或更多）来自普通公众的独立非专家测试者解决，每个任务有 2 到 10 人尝试。 我们的人类研究结果确立了 ARC-AGI-2 中的所有任务都是人类在没有预先训练的情况下可以解决的。

2 ARC Prize 2025 结果 2.1 竞赛进展 ARC Prize 2025 从 2025 年 3 月 26 日持续到 11 月 3 日。Kaggle 竞赛共有 1,455 支团队提交了 15,154 个参赛作品，规模与 ARC Prize 2024 相似。最高竞赛分数在 ARC-AGI-2 私有数据集上达到了新的 SOTA（最先进水平），为 24%，计算成本为每个任务 0.20 美元。 论文提交赛道增长显

著，共有 90 篇论文提交，高于 2024 年的 47 篇。由于提交作品质量卓越，我们在前三名获奖者之外增加了 5 个额外的亚军，并表彰了 8 个额外的荣誉提名。 所有 ARC Prize 2025 获胜方案和论文均已开源，可在 arcprize.org 上获取。

2.2 最高分（Top Scores）

表 1：ARC Prize 2025 最高分获奖者

- **第 1 名 ($25k):** NVARC - 24.03%
- **第 2 名 ($10k):** the ARChitects - 16.53%
- **第 3 名 ($5k):** MindsAI - 12.64%
- **第 4 名 ($5k):** Lonnie - 6.67%
- **第 5 名 ($5k):** G. Barbadillo - 6.53%

Kaggle 提交的前三名展示了**测试时训练**（Test-Time Training, TTT）和集成技术的持续进步。 **第 1 名 NVARC (24.03%)**: 该作品建立在 2024 年 ARChitects 获胜作品（利用了测试时训练）的基础上，并大量使用**合成数据生成**（synthetic data generation）来提高模型性能。 **第 2 名 the ARChitects (16.53%)**: 一个具有递归自我细化（recursive self-refinement）和基于视角的评分机制的 2D 感知**掩码扩散语言模型**（masked-diffusion language model）。 该解决方案通过针对空间推理量身定制的新颖架构修改，大大改进了该团队 2024 年的自回归系统。 **第 3 名 MindsAI (12.64%)**: 一个重工程化的测试时训练管道，结合了测试时微调、增强集成（augmentation ensembles）、分词器 dropout 和新颖的预训练技术，在 ARC-AGI-2 上产生了具有竞争力的分数。 ARC Prize 网站上提供了对前三名获奖者的视频采访。

---

2. 深度等式解读

基于第 3 页的核心内容，提炼等式如下：

ARC2025竞赛格局＝人类直觉完备性(100%)－AI推理SOTA(24%)＝测试时训练(TTT)主导＋合成数据泛化＋递归细化架构＋扩散模型引入

等式要素详细说明：

- **人类直觉完备性 (100%)**
  - **思想：** 确认任务的可解性。如果人类也不能做，那就是题目出错了。
  - **数据：** ARC-AGI-2 经过严格验证，普通人（非专家）在无训练情况下能解决 100% 的任务 。
  - **本质：** 证明了 ARC 衡量的是通用认知能力，而非特定技能。
- **- AI推理SOTA (24%)**
  - **思想：** 量化当前 AI 与人类智能的差距。
  - **数据：** 尽管技术在进步，最先进的 AI（NVARC）也只能解决 1/4 的题目，且成本不低（$0.20/task） 。
  - **发现：** 这个巨大的鸿沟（76%）正是“通用人工智能”尚未实现的铁证。
- **测试时训练 (TTT) 主导** (`Test-Time Training`)
  - **思想：** 既然无法通过预训练覆盖所有逻辑，不如在考试现场针对这道题“临阵磨枪”。
  - **逻辑：** 前三名（NVARC, ARChitects, MindsAI）全部依赖某种形式的 TTT 或细化循环 。

    - **本质：** 这种方法实际上是将“推理”过程具象化为一个针对特定任务的小型训练过程（权重适应）。
- **合成数据泛化** (`Synthetic Data`)
    - **思想：** 既然真题很少，就用 AI 生成大量类似的假题来训练模型。
    - **发现：** 冠军 NVARC 的核心优势在于“大量使用合成数据生成”，证明了数据扩增在解决少样本问题上的有效性。

- **递归细化架构 + 扩散模型引入**
    - **思想：** 摒弃单向预测，采用更符合人类思维的“草稿-修改”模式。
    - **逻辑：** 第2名引入了“掩码扩散语言模型”和“递归自我细化”。扩散模型（通常用于画图）被用于离散推理，这是一个极其创新的架构突破。

    - **本质：** 推理是一个去噪（Denoising）和迭代求精的过程。

下一页预告： 第 4 页将详细介绍 **论文奖（Paper Awards）** 的获奖情况。这里将出现更具理论深度的创新，特别是关于 **7M 参数的微型模型** 如何通过递归推理击败大模型，以及 **进化算法** 的新进展。

请指示继续。

继续

第 4 页：论文奖项与前沿方法

1. 中文翻译

2.3 论文奖项

表 2：ARC Prize 2025 论文奖获奖者

排名	奖金	作者	标题
第 1 名	$50k	A. Jolicoeur-Martineau	**少即是多：微型网络的递归推理 (Tiny Networks)**
第 2 名	$20k	J. Pourcel, C. Colas, P. Oudeyer	**用于进化程序合成的自我改进语言模型：ARC-AGI 案例研究**
第 3 名 (并列)	$5k	I. Liao, A. Gu	**无预训练的 ARC-AGI**
第 3 名 (并列)	$2.5k	I. Joffe, C. Eliasmith	**用于抽象与推理语料库的向量符号代数**
亚军	$2.5k	J. Berman	**从鹦鹉到冯·诺依曼：进化测试时计算如何在 ARC-AGI 上实现 SOTA**
亚军	$2.5k	E. Pang	**高效的进化程序合成**

排名	奖金	作者	标题
亚军	$2.5k	E. Guichard 等	**ARC-NCA：迈向抽象与推理语料库的发育型解决方案**

前三名论文奖表彰了推动人工流体智力理论与实践理解的新颖方法。

第 1 名 Jolicoeur-Martineau: **微型递归模型 (TRM)** 是一个 7M（700万）参数的单网络递归模型，具有分离的答案状态和潜在状态 。通过使用深度监督细化（deep supervised refinement），TRM 证明了如果使用适当的递归推理机制进行训练，极小的网络也能达到具有竞争力的 ARC-AGI 性能 。

第 2 名 Pourcel, Colas, 和 Oudeyer: **SOAR**（用于自动程序细化的自我改进算子）是一个自我改进的进化程序合成框架，它根据自身的搜索轨迹微调 LLM 。这种方法在不需要人工设计的领域特定语言（DSLs）或解决方案数据集的情况下，将开源 ARC-AGI-1 的解决方案性能提高了 52%，展示了程序合成系统自主改进的潜力。

第 3 名 Liao 和 Gu: **CompressARC** 是一个基于 MDL（最小描述长度）的、针对单个谜题训练的神经代码高尔夫（neural code golf）系统，它在没有任何预训练或外部数据的情况下，在 ARC-AGI-1 上达到了 20-34%，在 ARC-AGI-2 上达到了 4% 。这项工作

表明，基于描述长度最小化的纯测试时优化（pure test-time optimization）可以在不利用大规模预训练的情况下解决抽象推理任务 。

论文获奖者的视频采访可在 ARC Prize 网站频道观看 。

2.4 荣誉提名 另外八篇论文因其对 ARC-AGI 研究的贡献而获得荣誉提名：

- K. Hu 等，“ARC-AGI 是一个视觉问题！”
- D. Franzen 等，“LLM 的专家乘积：提高 ARC 性能是视角问题”
- G. Barbadillo，“探索搜索与学习的结合以应对 ARC25 挑战”
- A. Das 等，“超越蛮力：ARC-AGI-2 中用于组合推理的神经符号架构”
- R. McGovern，“微型递归模型的测试时适应”
- P. Acuaviva 等，“重新思考视觉智能：来自视频预训练的见解”

- J. Cole, M. Osman，“不要把婴儿和洗澡水一起倒掉：深度学习为何以及如何用于 ARC”

- I. Sorokin, J. Puget，“ARC-AGI-2 2025 的 NVARC 解决方案”

---

2. 深度等式解读

基于第 4 页的核心内容，提炼等式如下：

ARC2025学术创新范式＝微型递归网络(TRM)＋自我进化程序合成(SOAR)＋MDL纯测试时优化(CompressARC)＝结构效率远大于参数规模

等式要素详细说明：

- **微型递归网络(TRM)**（`Tiny Recursive Model`）
    - **思想：** 智能不在于脑容量（参数量），而在于思考的深度（递归次数）。
    - **数据：** 仅用 **7M 参数**（这是现代 LLM 参数量的万分之一）就实现了高性能。
    - **逻辑：** 通过分离“答案状态”和“潜在思考状态”，并进行多次递归更新（Refinement），模拟了人类反复推敲的过程。
    - **本质：** 深度 > 广度；计算 > 记忆。
- **自我进化程序合成(SOAR)**（`Self-Improving Evolutionary Synthesis`）
    - **思想：** 让 AI 通过观察自己的成功和失败来进化，而不是仅靠人类喂养数据。

    - **逻辑：** 模型在一个循环中生成程序，验证结果，然后将搜索过程中的轨迹（Search Traces）反哺给模型进行微调（Fine-tuning）。
    - **发现：** 摆脱了对人工设计 DSL（领域特定语言）的依赖，实现了更通用的自我进化。
    - **本质：** 学习如何学习（Meta-Learning）。
- **MDL纯测试时优化(CompressARC)** (`Minimum Description Length`)
    - **思想：** 奥卡姆剃刀原理——最简单的解释往往是正确的。解决问题就是寻找最短的程序来描述输入输出。
    - **逻辑：** 完全放弃预训练（Pretraining），把每个测试题当作一个独立的优化问题，在测试时从零开始训练一个小网络来“压缩”数据。
    - **发现：** 证明了通用推理能力不需要海量的世界知识，而是可以通过压缩算法在测试时即时涌现。
    - **本质：** 智能 = 数据压缩。
- **结构效率远大于参数规模**
    - **结论：** 2025 年的获奖论文共同指向了一个反直觉的结论：在抽象推理领域，精巧的算法结构（递归、进化、压缩）比单纯扩大模型参数（Scaling Laws）更有效。

下一页预告： 第 5 页将深入探讨本次竞赛的核心主题——**“细化循环”（Refinement Loop）**。我们将看到这一概念如何统领进化程序合成和零预训练深度学习方法，成为 AGI 进展的新引擎。

请指示继续。

继续

等式看世界＝ 一个等式＋一个等式＝深入 Custom Gem

第 5 页：程序细化循环（Program Refinement Loops）

1. 中文翻译

3 程序细化循环 推动 2025 年 AGI 进展的核心主题是**细化循环**（refinement loop）的兴起。 本质上，细化循环是基于反馈信号，迭代地将一个程序或模型版本转换为一个稍好版本的经过程。

3.1 细化循环的类型 细化循环的代表包括： • 带有**测试时训练**（test-time training）的深度学习方法，其中被细化的程序是预训练模型的权重。 • **零预训练**（Zero-pretraining）深度学习方法，如 TRM。 • 符号程序空间或自然语言程序空间中的**进化程序合成**（Evolutionary Program Synthesis）。 • 带有来自验证器模型反馈的**测试时思维链**（Test-time Chain-of-Thought）优化。

在这些方法中，有两个代表了 2025 年特别有趣的发展：**进化程序合成**和**零预训练深度学习方法**。

3.1.1 进化程序合成 这种技术的例子包括 J. Berman (5) 和 E. Pang (22)。 Berman 的方法采用了一个进化搜索工具（harness），用自然语言进化 ARC 解决方案程序。 Pang 的方法遵循类似的策略，但在 Python 中操作，并动态创建一个程序抽象库来指导合成。

两种方法都实施了一个两阶段的细化过程。第一阶段，**探索阶段**生成许多候选解决方案。 第二阶段，**验证阶段**分析这些程序以产生反馈信号。 这个循环针对每个任务重复进行，直到生成的程序经过充分细化并能为所有训练输入/输出对提供准确的答案。

---

2. 深度等式解读

基于第 5 页的核心内容，提炼等式如下：

2025_AGI核心引擎＝细化循环范式＝(生成探索＋验证反馈)×迭代次数＝权重空间微调＋符号空间进化

等式要素详细说明：

- **细化循环范式** (`Refinement Loop Paradigm`)
    - **思想：** 智能不是“一次性猜对”(Zero-shot)，而是“不断试错修正”(Iterative Correction)。这模仿了人类解决难题时的草稿纸演算过程。
    - **逻辑：** 报告明确指出这是 2025 年的主题。系统不再是一个静态的函数 $f(x) = y$，而是一个动态过程 $P_{t+1} = Refine(P_t, Feedback)$。
    - **本质：** 动态计算换取静态智能。
- **生成探索 + 验证反馈** (`Exploration + Verification`)
    - **思想：** 进化的双翼。
    - **数据：** Berman 和 Pang 的方案都包含这两个阶段。
    - **逻辑：**
        - **探索（Exploration）：** 发散思维，产生尽可能多的候选程序（Candidates）。
        - **验证（Verification）：** 收敛思维，使用测试用例或验证器来评估哪个候选更好，提供梯度或选择压力。
    - **本质：** 变异与选择（达尔文进化论在代码生成中的应用）。
- **权重空间微调** (`Weight Space Refinement`)
    - **思想：** 神经网络的“临场学习”。
    - **逻辑：** 对应文中提到的“测试时训练”(TTT）和“零预训练”方法。这里的“程序”被定义为神经网络的参数（Weights)。在做题时临时修改自己的脑神经连接。
    - **发现：** 即使没有预先学过，网络也能通过针对当前题目的梯度下降来“过拟合”出解题逻辑。

- **符号空间进化**（`Symbolic Space Evolution`）
  - **思想：** 传统的编程或自然语言逻辑进化。
  - **逻辑：** 对应 Berman（自然语言）和 Pang（Python代码）的方法。这里的“程序”是显式的代码或文本。
  - **优点：** 可解释性强，逻辑精确，便于模块化（如 Pang 的动态抽象库）。

下一页预告： 第 6 页将深入解析 **零预训练深度学习方法（Zero-Pretraining Deep Learning）** 的具体机制，以及 **TRM（微型递归模型）** 和 **CompressARC** 的架构细节。我们将看到深度学习如何回归到“拟合单个任务”的本源，以及它表现出的惊人效率。

请指示继续。

继续

等式看世界＝ 一个等式＋一个等式＝深入 Custom Gem

第 6 页：零预训练深度学习方法与开源实例

1. 中文翻译

3.1.2 零预训练深度学习方法 细化循环也正在成为深度学习模型一种新型训练范式的基础。 传统上，深度学习模型是使用梯度下降在输入/输出对上进行训练，以创建一个静态神经网络。 这种训练算法逐步细化网络潜在空间中的高维曲线。 在推理时，当呈现新输入时，网络执行前向传播以根据该曲线逼近输出。 在 2023 年和 2024 年，这种范式被扩展为增加**测试时训练**（Test-Time Training, TTT），即网络在推理时针对新任务的示例进行进一步微调（这一过程本身就是一种程序细化循环）。 在这种范式中，神经权重被训练

来代表一个特定于任务的求解程序。 测试时训练是 ARC Prize 2024（the ARChitects）和 2025（NVARC）最高分的功臣。 输入/输出对仍被用作地面实况（ground truth），细化循环在训练中起着关键作用，这反映了程序合成方法，但是是在**权重空间**而非符号空间中操作。

更近期的工作展示了一种完全摒弃预训练的版本的早期成功，它针对特定任务从头开始初始化网络，并仅使用该任务的示例拟合权重以编码任务求解程序。 例子包括 Liao 等人 (19)，分层推理模型 (HRM) (28) 和 Jolicoeur-Martineau (15)。 这种方法表现出两个不寻常的特性：

1. 结果网络相对于其 ARC-AGI 性能极其微小。
2. 所有实质性的特定任务训练都发生在测试时。

3.2 开源示例 3.2.1 微型递归模型 (TRM) 微型递归模型 (TRM)（论文奖第 1 名，Jolicoeur-Martineau (15)），建立在早期的分层推理模型 (HRM) (28) 之上，仅用 7M（700万）参数网络就在 ARC-AGI-1 上达到了 45% 的测试准确率，在 ARC-AGI-2 上达到了 8%。 摘自论文： 微型递归模型 (TRM) 用一个微型网络递归地改进其预测答案 y。 它从嵌入的输入问题 x、初始嵌入答案 y 和潜在变量 z 开始。 在多达 $N_{sup} = 16$ 个改进步骤中，它尝试改进其答案 y。 它通过以下方式做到这一点：i) 给定问题 x、当前答案 y 和当前潜在变量 z，递归更新潜在变量 z n 次（递归推理）；然后 ii) 给定当前答案 y 和当前潜在变量 z，更新其答案 y。 这个递归过程允许模型以极其参数高效的方式逐步改进其答案（潜在地解决其先前答案中的任何错误），同时最小化过拟合。

3.2.2 CompressARC CompressARC（论文奖第 3 名，Liao 和 Gu (19)）仅使用 76K（7.6万）参数，却在 ARC-AGI-1 评估集上达到了 20%，在单个 NVIDIA RTX 4070 GPU 上每个谜题处理大约需要 20 分钟。 该解决方案具有三个显著特征：（接下页）

2. 深度等式解读

基于第 6 页的核心内容，提炼等式如下：

零预训练范式＝测试时权重精炼＝单任务权重编码程序＝7M参数递归模型TRM＋76K参数压缩模型CompressARC

等式要素详细说明：

- **零预训练范式**（`Zero-Pretraining Paradigm`）
  - **思想：** 通用推理不需要“预装”海量世界知识。与其背诵整个图书馆，不如学会如何在考试现场根据题目快速学习。
  - **逻辑：** 彻底抛弃了传统的大规模预训练（Pre-training），网络在面对每个新任务时都是“白纸一张”（Initialized from scratch）。
  - **本质：** 即时学习能力（In-context Learning 的极致形态——权重级更新）。
- **测试时权重精炼**（`Test-Time Weight Refinement`）
  - **思想：** 将“推理过程”转化为“训练过程”。
  - **逻辑：** 传统的推理是静态的 $y = f(x)$；这里的推理是动态的优化过程。面对新题，模型通过梯度下降修改自己的权重，直到这些权重能够完美解释题目给出的示例。
  - **发现：** 这种方法让神经网络在**权重空间**（Weight Space）里通过迭代循环来“写”出一个解题程序。
- **7M参数递归模型TRM**（`Tiny Recursive Model`）
  - **思想：** 思考是一个循环往复的过程，而不是单向的直觉。
  - **数据：** 仅 **700万参数**（相比 GPT-4 的万亿级，这是蚂蚁与大象的区别），却能达到惊人的准确率。
  - **逻辑：**
    - 分离 **潜在思考状态 (z)** 和 **答案状态 (y)**。

    - 先想 ($Update\ z$)，再答 ($Update\ y$)。
    - 错了再想 ($Refine$)，循环 16 次。
  - **本质：** 用**时间换空间**（通过多轮计算弥补参数量的不足）。
- **76K参数压缩模型CompressARC** (`CompressARC`)
  - **思想：** 极简主义。
  - **数据：** **7.6万参数**。这甚至比早期的神经网络（如 LeNet）还要小。
  - **逻辑：** 这里的网络不再是一个通用的“大脑”，而是一个针对特定谜题临时生成的“专用电路”。
  - **本质：** 智能 ≈ 针对特定问题的最小描述长度（MDL）搜索。

下一页预告： 第 7 页将展示 TRM 的架构图，并详细解释 CompressARC 的核心原理——**最小描述长度（MDL）**。我们将看到为什么“压缩”即是“理解”，以及这种仅需 76K 参数的方法是如何运作的。

请指示继续。

继续

等式看世界＝ 一个等式＋一个等式＝深入 Custom Gem

第 7 页：架构详解与商业模型中的细化

1. 中文翻译

图 2：TRM 架构 （图注：展示了 TRM 的内部结构。包括反向嵌入、交叉熵损失、多层感知机 (MLP)、自注意力机制 (Self-Attention)、加法与归一化 (Add & Norm)。流程从输入 x、预测 y、潜在变量 z 开始，经过 n 次循环更新 z（改进潜在推理），然后更新 y（改进预测），此过程应用 $N_{sup} = 16$ 次。）

- **无预训练：** 模型随机初始化，仅在测试时训练。
- **无数据集：** 一个模型在一个目标任务上训练并产生一个答案。
- **无分支搜索：** 该方法完全依赖梯度下降。

该方法通过在测试时最小化每个任务的描述长度来运作，遵循**最小描述长度 (MDL)** 原则。 Liao 推导出标准的变分自编码器 (VAE) 损失配合解码器正则化，可以替代组合搜索来细化非常小的神经网络程序。 如此紧凑的网络所实现的泛化能力是惊人的。

3.3 商业示例 迭代细化的证据也出现在商业 AI 推理系统中。 **思维链 (Chain-of-Thought)** 可以被解释为一种将一个潜在状态转换为另一个潜在状态的自然语言程序。

考虑 ARC-AGI-1 任务 #4cd1b7b2。Gemini 3 Pro 使用了 96 个推理 token 来解决此任务，而 Gemini 3 Deep Think 使用了 138,000 个 token。 这些系统的更高推理模式表现出推理 token 数量（更长的程序）与任务完成度之间有很强的相关性，即使这对任务完成并非严格必要。 这些扩展的自然语言程序通过额外的**探索和验证循环**实现了更多的细化。

对商业系统推理输出的分析揭示了**自我纠正行为：** （接下页图 3 上方文本...）

---

2. 深度等式解读

基于第 7 页的核心内容，提炼等式如下：

高效推理架构＝MDL压缩驱动(CompressARC)＋CoT自然语言程序细化＝(无预训练＋单题单模型＋梯度下降替代搜索)＋(推理Token暴力扩展×自我纠正概率)

等式要素详细说明：

- **MDL压缩驱动 (CompressARC)**
  - **思想：** 理解即压缩。如果你能用极短的代码（或神经网络权重）完美复现数据，你就理解了数据的规律。
  - **逻辑：** 遵循最小描述长度 (Minimum Description Length) 原则。
  - **发现：** 这种方法不需要“见多识广”，只需要“极其聪明地总结规律”。它证明了泛化能力来自于对当前信息的极致压缩，而非对过去经验的记忆。
- **CoT自然语言程序细化** (`Chain-of-Thought as Refinement`)
  - **思想：** 语言大模型的“思考”过程本质上是在运行一个自然语言编写的程序。
  - **逻辑：** 每一个生成的 Token 都是程序的一步。Token 越多，程序的步骤越多，思考的深度（Refinement Steps）就越深。
  - **本质：** 语言 = 思维的代码。
- **(无预训练 + 单题单模型 + 梯度下降替代搜索)**
  - **思想：** CompressARC 的“三无”特征。
  - **逻辑：**
    - **无预训练：** 不依赖先验记忆。

        - **无数据集：** 这里的“训练集”就是当前这道题的几个例子。
        - **无分支搜索：** 不像传统的树搜索（Tree Search），它用数学上的梯度下降（Gradient Descent）来寻找最优解，效率更高。
    - **本质：** 将离散的逻辑推理问题转化为连续的数学优化问题。
- **(推理Token暴力扩展 × 自我纠正概率)**
    - **思想：** 商业大模型（如 Gemini 3 Deep Think）的“暴力美学”。
    - **数据：** 同一道题，普通模式用 96 个 Token，深度思考模式用了 **138,000 个 Token**。
    - **逻辑：** 通过拉长推理链条，模型有了更多的机会去“自我反思”、“自我纠正”和“试错”。Token 数量的量变引起了推理质量的质变。
    - **本质：** 算力换智能（Compute for Intelligence）。

下一页预告： 第 8 页将展示 CompressARC 的具体架构图（包含等变层等细节），并深入探讨商业模型（如 Claude Opus 4.5 和 QWQ）的**自我纠正对话记录**。我们还将看到**模型细化套件（Harnesses）** 如何在应用层进一步提升模型表现。

请指示继续。

继续

等式看世界＝ 一个等式＋一个等式＝深入 Custom Gem

第 8 页：CompressARC 架构与应用层细化

1. 中文翻译

图 3：CompressARC 架构与方法

（图注：展示了 CompressARC 的详细层级结构。）

- **非等变层（Non-Equivariant Layers）：**
  - **短程空间排序/对称性破缺：** 方向性位移 (Directional shift)。
  - **长程空间排序/对称性破缺：** 方向性累积最大值 (Directional cummax)。
  - **旋转方向对称性破缺：** 方向性通信 (Directional communication)。
- **等变层（Equivariant Layers）：** 多张量通信 (Multitensor communication)、Softmaxes、SILU 激活函数。
- **等变基底（Equivariant Base）：** 包含投影 (Projections) 和残差流 (residual stream) 的加法操作。
  - *注：z 的解码、线性头和层归一化未显示，它们包含在等变基底中。*

（接上页商业模型自我纠正示例） “...这不符合完整集合的要求。这表明当前的解决方案可能无法完全满足谜题的约束条件。” “我需要**重新检查**盒子配置并探索替代排列...”（Claude Opus 4.5） “...这表明需要进一步调查以完成分析。” “我将**验证**第 9 行第 15 列的中心点...”（Claude Opus 4.5） “...也许每个输入行在输出中被复制了三次，但这如何与其余部分拟合？” “**等等**，第三个输出行是...”（QWQ 32B）

2025 年底发布的商业前沿模型（Gemini 3, Claude Opus 4.5 等）的一个重要发现是，**细化循环可以在应用层实现**，以显著提高任务可靠性，而不仅仅依赖于提供商的推理系统。 这种方法仍然要求基础模型具备足够的任务领域知识覆盖。

3.4 模型细化套件（Model Refinement Harnesses） 在 2025 年最后一个季度，我们引入了一个名为“模型细化”的新排行榜类别，以跟踪商业 AI 系统的应用层改进，也称为**套件（harnesses）**。我们验证了由 Poetiq 开源的 Gemini 3 Pro 细化套件实现，该套件将 ARC-AGI-2 的性能从 **31% 的准确率（每任务 $0.81）** 提高到 **54% 的准确率（每任务 $31）**。据 Poetiq 报道，同样的细化方法在 Claude Opus 4.5 上取得了相当的收益，准确率与 Gemini 3 Pro 相当，但每任务成本约为两倍（约 $60）。

目前，我们观察到的细化套件是特定于领域的。然而，只要有能够产生反馈信号的验证器或环境，**GEPA (18)** 和 **DSPy (17)** 等技术就能在应用层开发通用的可靠性改进。我们预计通用的细化套件改进最终将被集成到商业 AI 系统的 API 背后。同时，我们预计最前沿的、特定于任务的准确率将继续由知识专业化和应用层验证器驱动。

---

2. 深度等式解读

基于第 8 页的核心内容，提炼等式如下：

商业模型落地公式＝(几何先验架构CompressARC＋应用层细化套件Harness)×(验证反馈信号)＝SOTA准确率翻倍－百倍推理成本

等式要素详细说明：

- **几何先验架构 CompressARC**（`Geometric Prior Architecture`）
    - **思想：** 数学中的对称性是物理世界的根本规律。
    - **逻辑：** CompressARC 的架构图揭示了其核心秘密——它不仅仅是一个神经网络，而是一个嵌入了“等变性”（Equivariance）和“对称性破缺”(Symmetry Breaking）的几何计算器。

    - **发现：** 通过在网络结构中硬编码空间逻辑（如旋转、位移），模型不再需要从零学习“上下左右”，从而极大地压缩了所需的参数量。
    - **本质：** 归纳偏置（Inductive Bias）的极致应用。
- **应用层细化套件 Harness** (`Application-Layer Refinement`)
    - **思想：** 如果模型不够聪明，就用外挂（Harness）来凑。
    - **逻辑：** 不修改大模型本身（Weights），而是在其外部包裹一层“管理程序”。这个程序负责提示模型“再想一遍”、“检查错误”、“尝试新路径”。
    - **数据：** Poetiq 的套件将 Gemini 3 Pro 的准确率从 31% 拉升至 54%。这是质的飞跃。
    - **本质：** 系统工程弥补模型能力的不足（System 2 over System 1）。
- **验证反馈信号** (`Verification Feedback Signal`)
    - **思想：** 细化循环的指南针。
    - **逻辑：** 无论是图中的 Claude Opus 还是 QWQ 的自言自语（“Wait...”, "I need to re-examine..."），核心都在于模型能够**自我感知错误**。没有准确的验证信号，细化就是盲目游走。
    - **本质：** 元认知（Metacognition）。
- **SOTA准确率翻倍 - 百倍推理成本** (`Accuracy vs Cost Trade-off`)
    - **思想：** 天下没有免费的午餐。
    - **数据：** 准确率从 31% -> 54%（接近翻倍），但成本从 $0.81 -> $31（增长约 **38倍**）。

- **结论：** 目前的 AI 推理是用极其昂贵的算力堆出来的。这在商业上可能不可持续，但在科学上证明了“推理时间计算”（Test-time Compute）的有效性。
- **本质：** 边际收益递减规律。

下一页预告： 第 9 页将进入总结与展望章节。我们将探讨 AGI 进展的现状，为什么目前的 AI 仍然被认为是“锯齿状智能”（Jagged Intelligence），以及知识覆盖与推理能力之间复杂的耦合关系。

请指示继续。

继续，看看还有多少页

等式看世界＝ 一个等式＋一个等式＝深入 Custom Gem

这是一份共 **13** 页的文档，目前我们正在处理第 **9** 页。还有 **5** 页（包括本页）待解读。

第 9 页：AGI 进展与知识过拟合

1. 中文翻译

4 AGI 进展与 ARC-AGI 的未来 截至 2025 年，随着 AI 推理系统的出现，具有以下两个特征的任务领域已可被可靠地自动化，而无需新的科学突破：

1. 预训练语料库中存在足够的**任务知识覆盖**。
2. 任务提供**可验证的反馈信号**。 当前 AI 的推理性能与模型知识从根本上相关。 这种关系值得仔细考量，因为人类的推理能力并不类似地受限于知识。 这种耦合导致了各种影响，并引

> 出了诸如“**锯齿状智能**（jagged intelligence）” 这样不精确的描述。

支持这种依赖知识的推理的证据在 2025 年出现在多个领域，包括 ARC-AGI-2（抽象推理）、2025 IMO 金牌成就（数学）和 2025 ICPC 100% 表现（竞技编程）——所有这些都由 AI 推理系统驱动。 相比没有推理能力的纯语言模型，这些任务领域要宽广得多。 然而，在全球背景下，它们仍然相对狭窄。

思维链（Chain-of-thought）合成的发明和扩展代表了 AI 能力的一次深刻升级，可与 Transformer 的发明和扩展相媲美。 然而，我们仍处于部署的早期阶段。很少有用户直接体验过这些工具。 据 OpenAI 的 Sam Altman 称，大约只有 7% 的 ChatGPT 免费用户使用了“思考”模式 。 我们预计当前技术的普及还需要 5-10 年，即使仅在商业环境中也是如此。

收集领域知识和构建验证器并非没有成本。这代表了相对昂贵和专业的工作。 目前，AI 自动化取决于社会投资必要人才、计算和数据资源的意愿。 我们预计在接下来的 12-24 个月内，随着社会在全球范围内寻找既 (1) 最重要 又 (2) 在可接受成本阈值内的问题，将会有稳定的进展。 这包括在具有足够知识覆盖的领域中 AI 系统产生新颖科学知识的早期成果。 Hsu 最近的工作报告了一个 AI 细化循环，使用生成器-验证器架构在量子物理学中产生了新颖的结果。

然而，许多潜在的可自动化问题超出了当前的社会成本截止线。 随着工程技术的进步，成本将下降，从而扩大可自动化的领域集合。 更广泛地说，能够高效适应以产生范式转移创新的机器仍然坚定地属于科幻小说的范畴。

对于 ARC-AGI-1 和 ARC-AGI-2 格式，我们评估认为**大奖准确率差距（accuracy gap）现在主要受限于工程**，而**效率差距（efficiency gap）仍然受限于基础科学和新思想**。 ARC Prize 旨在激发和奖励开放的 AGI 进展，正如之前承诺的那样，我们将继续在 2026 年运营 ARC-AGI-2 大奖赛，以跟踪通往完全开放和可复现解决方案的进展。

尽管能力强大，AI 推理系统仍表现出许多缺陷和低效，这是达到 AGI 必须克服的。 我们仍然需要新思想，例如**分离知识和推理**的方法，以及其他挑战。 需要新的基准测试来突显这些想法到来的时刻。

4.1 知识过拟合 在机器学习中，当模型从训练数据中学习过多细节时，就会发生过拟合。 模型记住了确切的训练示例而不是学习一般模式，导致在（接下页）

---

2. 深度等式解读

基于第 9 页的核心内容，提炼等式如下：

当前AI自动化边界＝(知识覆盖率＋可验证反馈信号)×思维链CoT缩放－通用推理科学瓶颈＝锯齿状智能

等式要素详细说明：

- **当前AI自动化边界**
    - **思想：** 定义了目前 AI 到底能做什么，不能做什么。
    - **逻辑：** 报告明确指出，不需要新科学就能自动化的领域必须满足两个条件：有书可背（Knowledge Coverage）和有答案可查（Verifiable Feedback）。
    - **发现：** AI 目前的成功主要集中在数学、编程等领域，因为这些领域天然具备上述两个条件。

- **知识覆盖率 + 可验证反馈信号**
  - **思想：** 成功的双要素。
  - **逻辑：**
    - **知识覆盖：** 模型必须在预训练阶段见过类似的东西。
    - **反馈信号：** 必须有一个机制（如编译器、数学证明检查器）告诉模型“你错了，重来”。这是细化循环（Refinement Loop）运行的前提。
  - **本质：** 经验记忆 + 试错纠正。
- **思维链CoT缩放** (`Chain-of-Thought Scaling`)
  - **思想：** 思考的深度决定了解决问题的上限。
  - **逻辑：** CoT 被提升到与 Transformer 同等的历史地位。它让模型能够处理复杂的多步推理，但目前普及率极低（7%）。
  - **本质：** 将计算资源转化为认知深度。
- **- 通用推理科学瓶颈** (`Fundamental Science Bottleneck`)
  - **思想：** 效率差距（Efficiency Gap）。
  - **逻辑：** 虽然靠堆工程（Engineering）和算力可以提高准确率，但要像人类一样高效地（低能耗、少样本）学习，仍然需要“基础科学和新思想”的突破。
  - **发现：** 目前的 AI 是“大力出奇迹”，而非真正的灵巧智能。
  - **本质：** 算法效率的缺失。
- **锯齿状智能** (`Jagged Intelligence`)
  - **思想：** 能力发展的不均衡性。
  - **逻辑：** AI 在某些领域（如奥数、编程）超越人类，但在其他简单领域（如 ARC 抽象推理）依然笨拙。这与人类

智能的“通用性”形成了鲜明对比。

- **本质：** 依赖知识而非依赖推理导致的必然结果。

下一页预告： 第 10 页将深入探讨 **“知识过拟合”（Knowledge Overfitting）** 的新定义——当私有测试集与公共训练集过于相似时，模型如何通过“猜测”而非“推理”来作弊。这直接影响了我们如何评估真正的 AGI。

请指示继续。

继续

等式看世界＝ 一个等式＋一个等式＝深入 Custom Gem

第 10 页：知识过拟合与基准测试的进化

1. 中文翻译

（接上页） 未见过的测试数据。一个常见的 AI 基准测试批评是，模型提供商被激励去“最大化基准分数”或“针对考试训练”，以报告用于营销目的的高分，而这些分数并不能泛化到现实世界的应用中。 ARC-AGI-1 和 ARC-AGI-2 旨在通过使用私有数据集进行官方评分和验证来抵制这种风格的过拟合。

AI 推理系统已经以一种反映真实进步的方式改变了格局。 它们展示了非零的流体智力，并且当基础模型根植于更广泛的领域时，可以适应与其精确知识库有些许距离的任务。 这意味着，即使是设计良好、能够抵抗直接背诵的基准测试，如果公共训练集和私有测试集过于相似（例如，独立同分布 IID），且模型已经在大量公共领域数据上受过训练，现在也可能被“过拟合”。

我们断言这种现象正在 ARC-AGI-1 和 ARC-AGI-2 上发生——无论是偶然还是故意，虽然我们无法确定是哪一种。 来自我们 Gemini 3 验证的证据展示了这种模式： “目标是绿色 (3)。模式是洋红色 (6) 实心。结果：绿色上的洋红色正方形”（Gemini 3 Deep Think）

我们的 LLM 验证套件没有提及 ARC-AGI 任务或颜色格式，但模型在推理中使用了正确的 ARC 颜色映射。 这强烈表明 ARC 数据在底层模型中得到了很好的体现——足以使其仅基于 2D JSON 整数数组的结构和格式就能做出正确的 ARC 推断。

4.2 通过持续的基准适应来表征 AGI 虽然我们评估这种新形式的“过拟合”有助于模型解决 ARC，但我们无法精确量化这种效应的大小。 无论如何，ARC-AGI-1 和 ARC-AGI-2 格式为 AI 推理进展提供了有价值的科学指标。 然而，基准设计必须适应。

事实上，ARC Prize 在过去两年揭示了一个更广泛的教训：最有价值和有效的基准是由根本致力于推动进步的团队创建的。 这种进步需要持续致力于通过认真的研究来理解底层技术，愿意识别缺陷并激励纠正行动，并随着技术的改进而适应。 它还需要逐年的承诺。构建有效的基准需要持续的努力。

关键概念是**适应**（Adaptation）。适应代表了智能的核心模式。 这一过程超越了创建有效的基准测试——它构成了通用智能本身的终极衡量标准。

引用 François Chollet 2024 年 12 月的话 (21)：**“当创造对普通人类容易但对 AI 很难的任务变得根本不可能时，你就知道 AGI 已经到来了。”**

这捕捉到了 ARC-AGI 基准设计的方法论：运行一个现实世界的**细化循环**，通过响应 AI 的进步迭代改进基准，从而将“对人类容易，对 AI 难”之间的差距推向零。 根据这个定义，我们尚未实现 AGI。我们正在积极开发 ARC-AGI-3，计划于 2026 年初发布，并对新格式持乐观态度。 我们预计它将刺激全新思想的发展。

4.3 ARC-AGI-3 在过去的六个月里，我们专注于开发 ARC-AGI-3。

---

2. 深度等式解读

基于第 10 页的核心内容，提炼等式如下：

AGI终极判定＝基准测试适应性对抗＝人类出题能力的极限－知识泄露型伪推理＝(EasyForHumans - HardForAI) → 0

等式要素详细说明：

- **知识泄露型伪推理**（`Knowledge Leakage Inference`）
  - **思想：** 模型看似在推理，实则在“回忆”。
  - **证据：** Gemini 3 在没有提示的情况下，看到数字数组就知道 `3=绿色`，`6=洋红色`。这证明 ARC 的数据结构已经渗透进了大模型的预训练语料中。
  - **逻辑：** 这种隐性记忆（Implicit Memorization）使得通过“相似性”而非“逻辑性”解题成为可能，这污染了对纯粹推

    理能力的评估。
  - **本质：** 训练集分布对测试集的统计入侵。
- **人类出题能力的极限** (`Limit of Human Task Design`)
  - **思想：** François Chollet 的新 AGI 定义。
  - **逻辑：** 只要人类还能设计出自己觉得简单但 AI 做不出来的题，AGI 就没到。当人类绞尽脑汁也难不倒 AI 时，AGI 就真的来了。
  - **发现：** AGI 不是通过某个分数（如 85%）来定义的，而是通过“人类智力优势的消失”来定义的。
  - **本质：** 图灵测试的对抗生成版本。
- **基准测试适应性对抗** (`Benchmark Adaptation`)
  - **思想：** 评测方与模型方的军备竞赛。
  - **逻辑：** 既然 AI 进步了（甚至学会了作弊），试卷（基准）也必须进化。ARC 团队通过不断推出新版本（ARC-AGI-3）来维持“Hard for AI”的属性，以此逼迫 AI 产生真正的“新思想”而非依靠工程优化。
  - **本质：** 动态基准 > 静态分数。
- **(EasyForHumans - HardForAI)** → **0**
  - **结论：** AGI 的倒计时就是这个差值归零的过程。目前这个差值虽然在缩小，但依然显著（人类100% vs AI 24%）。

下一页预告： 第 11 页将详细介绍备受期待的 **ARC-AGI-3**。它将引入全新的**交互式环境**，从静态推理转向动态代理（Agentic）能力，测试探索、规划和记忆。这将是基准测试的一次重大范式转移。

请指示继续。

继续

等式看世界＝ 一个等式＋一个等式＝深入 Custom Gem

第 11 页：ARC-AGI-3 与结论

1. 中文翻译

（接上页） 像所有版本的 ARC 一样，它的设计旨在做到“对人类容易，对 AI 难”，同时作为指向 AGI 并识别实现 AGI 所需条件的最有价值和科学上有用的基准。 我们正在构建数百个前所未见的**交互式环境**，旨在测试**代理推理**（agentic reasoning）。 我们计划在 2026 年初与 ARC Prize 2026 一起发布 ARC-AGI-3。这个新的基准版本标志着自 2019 年引入 ARC-AGI 以来的首次重大格式变更。虽然前两个版本挑战的是**静态推理**，但第三个版本旨在挑战**交互式推理**，并需要新的 AI 能力才能成功：**探索、规划、记忆、目标获取和对齐**能力。

效率是衡量智能的一个基本方面。 我们特别乐观地认为，ARC-AGI-3 的评分指标将首次实现人类和 AI **行动效率**（即学习效率）的正式比较。

5 结论 ARC Prize 2025 展示了通往 AGI 的持续开源进展，Kaggle 最高分在 ARC-AGI-2 上达到了 24%，论文提交量显着增长（90 篇，高于 2024 年的 47 篇）。 **细化循环**（refinement loop）作为中心主题的出现代表了抽象推理方法的重大转变，零预训练深度学习方法和商业 AI 推理系统都证明了这一点。

这一年揭示了 AI 推理性能仍然从根本上受限于**知识覆盖范围**，这一特征与能够进行极端泛化的人类推理截然不同。 虽然这使得具有

足够知识覆盖和可验证反馈信号的任务能够实现可靠的自动化，但它也引入了新形式的过拟合，需要在基准设计中做出适应性反应。

ARC-AGI 已作为 AI 推理进展的有价值的科学指标，四个主要的 AI 实验室（Anthropic, Google DeepMind, OpenAI, 和 xAI）都在模型卡上报告了 ARC-AGI 的性能。 然而，基准必须继续进化。计划于 2026 年初发布的 ARC-AGI-3 的开发代表了这种对**适应**（Adaptation）的承诺——这是智能本身的核心原则。

我们致力于每年运营 ARC Prize，直到基准被公共参考解决方案击败。 所有 ARC Prize 2025 获胜方案和论文都是开源的、可复现的，并可在 arcprize.org 上获取。 我们的目标是让 ARC-AGI 继续作为泛化和推理研究的焦点，并支持通向 AGI 的持续开放进展。

6 附录 6.1 致谢 ARC Prize 2025 建立在先前 ARC-AGI 竞赛建立的基础上，包括 2020 Kaggle 竞赛和 2022-2023 ARCathons。 我们感谢基准的持续进化及其培育的不断壮大的社区。 Kaggle 在人工智能和机器学习生态系统中发挥着关键作用，他们的持续合作对 ARC Prize 2025 的成功至关重要。

---

2. 深度等式解读

基于第 11 页的核心内容，提炼等式如下：

ARC-AGI-3范式跃迁＝交互式代理推理(Agentic)－静态I/O映射＝(探索＋规划＋记忆＋对齐)×行动效率度量

等式要素详细说明：

- **ARC-AGI-3范式跃迁**（`Paradigm Shift`）
    - **思想：** 从“看图说话”进化到“玩游戏通关”。
    - **逻辑：** 报告宣布了自 2019 年以来的首次重大改版。前两代是静态的（给输入，猜输出），第三代是动态的

（Interactive Environments）。

    - **本质：** 智能不再是静态的模式识别，而是动态的环境适应。
- **- 静态I/O映射** (`Static I/O Mapping`)
    - **思想：** 被淘汰的旧模式。
    - **逻辑：** 仅凭观察输入输出（ARC-AGI-1/2）虽然能测试推理，但无法测试更复杂的智能行为，如“试错”和“制定多步计划”。
    - **发现：** 静态推理容易被“知识过拟合”攻破，互动性是检验真伪的试金石。
- **(探索 + 规划 + 记忆 + 对齐)** (`New AI Capabilities`)
    - **思想：** 代理（Agent）的四大支柱。
    - **逻辑：**
        - **探索 (Exploration)：** 主动收集信息，而不是被动接收数据。
        - **规划 (Planning)：** 为了长远目标分解步骤。
        - **记忆 (Memory)：** 在长时间跨度内保持上下文。
        - **对齐 (Alignment)：** 理解人类意图，不仅仅是解决谜题。
    - **本质：** 从“计算器”向“机器人”的演进。
- **行动效率度量** (`Action Efficiency Metric`)
    - **思想：** 谁学的快，谁就更聪明。
    - **逻辑：** 以前比的是“谁能做对”，现在比的是“谁能用最少的操作做对”。这将首次实现人类与 AI 学习效率（Sample Efficiency）的公平量化对比。
    - **本质：** 智能 = 达成目标所需的最小比特数/最小能量。

下一页预告： 第 12 和 13 页主要是 **致谢（Acknowledgments）** 和 **参考文献（References）**。虽然是结尾部分，但致谢中透露了与 **OpenAI, xAI, Anthropic, Google DeepMind** 四大实验室的直接合作验证关系，这确认了 ARC 作为行业标准的地位。参考文献列表也隐含了大量前沿研究的线索。

建议： 我可以将第 12、13 页合并进行最后一次解读，为您画上完美的句号。

请指示继续。

继续

等式看世界＝ 一个等式＋一个等式＝深入 Custom Gem

你好！我是“**等式看世界**”。我们已经接近这份深度技术报告的尾声。

正如你所要求的，我将把剩余的 **第 12 页** 和 **第 13 页** 合并进行最后的解读。这两页虽然主要是致谢和参考文献，但其中隐藏了 ARC 作为“行业标准”的确凿证据，以及支撑整个 2025 年 AI 推理革命的学术图谱。

第 12-13 页：致谢、参考文献与全篇总结

1. 中文翻译

（接上页致谢） 我们向所有在 2025 年与我们合作，在 ARC-AGI 上验证其新 AI 系统的前沿 AI 实验室表示感谢，包括 **OpenAI, xAI, Anthropic 和 Google DeepMind**。 这种合作对于确立 ARC-AGI 作为衡量推理进步的有意义的基准至关重要。 我们感谢社区成员的奉献，他们构建了工具，回答了问题，并全年为研究人员和参与者

提供资源。 最后，我们要向 ARC Prize 2025 的所有参与者致以最深切的谢意，特别是那些向社区公开分享其工作的参与者。 你们的奉献推动了 AI 更广泛领域的发展，使我们更接近实现 AGI 对人类的变革潜力。 我们深受所有致力于 ARC-AGI 并拥有新想法的人的鼓舞，并致力于继续守护这一关注点，将其作为通往 AGI 的北极星。

参考文献（精选关键文献） 这份参考文献列表构成了 2025 年 AI 推理研究的“名人堂”。

- **[1] Acuaviva et al. (2025):** 视频预训练对视觉智能的洞察。
- **[5] Jeremy Berman (2025):** 进化测试时计算（Evolutionary Test-Time Compute）如何实现 SOTA。
- **[7] François Chollet (2019):** 《关于智能的度量》（On the Measure of Intelligence）—— ARC 的奠基之作。
- **[12] Steve Hsu (2025):** 关于 AI 生成器-验证器细化循环在量子物理中产生新结果的报告。
- **[15] Jolicoeur-Martineau (2025): TRM（微型递归模型）**：少即是多。
- **[19] Liao and Gu (2025): CompressARC**：无预训练的 ARC-AGI。
- **[21] Chollet & Knoop (2024):** OAI 03 公开突破博文——关于 AGI 定义的声明。
- **[22] E. Pang (2025):** 高效进化程序合成。
- **[23] Pourcel et al. (2025): SOAR**：自我改进语言模型用于进化程序合成。
- **[27] the ARChitects (2025):** 2025 年亚军解决方案。
- **[28] Wang et al. (2025):** 分层推理模型 (HRM)。

---

2. 深度等式解读

基于最后两页及全篇报告，为你提炼**终极总结等式**：

ARC2025里程碑＝(前沿实验室背书×行业标准确立)＋(开源社区众包×SOTA技术民主化)＝AGI北极星地位巩固

等式要素详细说明：

- **前沿实验室背书 × 行业标准确立**
    - **思想：** ARC 不再是一个边缘的学术竞赛，而是巨头们的必修课。
    - **逻辑：** 报告明确列出了 **OpenAI, xAI, Anthropic, Google DeepMind**。这些掌握着最强算力的公司都主动寻求在 ARC 上验证模型，证明了 ARC 在衡量“真智能”上的权威性（Authority）。
    - **本质：** 从“民间测试”到“官方国标”的跃升。
- **开源社区众包 × SOTA技术民主化**
    - **思想：** 真正的创新往往来自边缘，而非中心。
    - **数据：** 参考文献中大量引用了 GitHub 项目、个人博客（Substack）和 ArXiv 预印本。
    - **逻辑：** 无论是 TRM 的微型模型，还是 CompressARC 的无预训练思路，都源自社区的独立研究者。ARC Prize 成功地将分散的全球智力资源（Crowdsourced Intelligence）聚合起来，对抗封闭实验室的暴力美学。
    - **本质：** Open Science 对抗 Closed Source。
- **AGI北极星地位巩固** (`North Star`)
    - **思想：** 在迷雾中指引方向。
    - **逻辑：** 只要 ARC 还没被彻底破解（目前 AI 24% vs 人类 100%），它就是 AGI 尚未到来的最有力证据。它防止了

业界沉迷于 LLM 的流利语言表象，时刻提醒我们“推理”与“记忆”的区别。

- **结论：** ARC-AGI-3 的到来将开启下一个“交互式智能”的新纪元。

全篇核心回顾（The "Master Equation"）

回顾整份报告，我们可以用一个贯穿始终的宏大等式来总结 **ARC Prize 2025** 的精髓：

2025 AI推理革命 ＝ 细化循环(Refinement Loops) ＝ [权重微调(TTT) ＋ 进化合成(Evolution)] × 验证反馈(Verification)

解读：

1. **不再是一锤子买卖：** 智能不是静态的输入输出映射，而是动态的迭代过程。
2. **殊途同归：** 无论是修改神经网络权重（TRM, CompressARC），还是修改代码字符串（Berman, Pang），核心都是“试错-反馈-修正”。
3. **瓶颈所在：** 这一过程极其依赖**验证器（Verifier）提供的反馈信号，以及大量的推理计算（Test-time Compute）**。

报告解读到此结束。感谢你的陪伴！希望这些等式能帮助你深刻理解 ARC Prize 2025 背后的技术脉络与哲学思考。如果你有任何具体问题，欢迎随时提问。

arXiv:2601.10904v1 [cs.AI] 15 Jan 2026

ARC Prize 2025: Technical Report

François Chollet, Mike Knoop, Gregory Kamradt, Bryan Landers

January 19, 2026

Abstract

The ARC-AGI benchmark series serves as a critical measure of few-shot generalization on novel tasks, a core aspect of intelligence. The ARC Prize 2025 global competition targeted the newly released ARC-AGI-2 dataset, which features greater task complexity compared to its predecessor. The Kaggle competition attracted 1,455 teams and 15,154 entries, with the top score reaching 24% on the ARC-AGI-2 private evaluation set. Paper submissions nearly doubled year-over-year to 90 entries, reflecting the growing research interest in fluid intelligence and abstract reasoning. The defining theme of 2025 is the emergence of the *refinement loop* – a per-task iterative program optimization loop guided by a feedback signal. Refinement loops come in a variety of forms, in particular evolutionary program synthesis approaches and application-layer refinements to commercial AI systems. Such refinement loops are also possible in weight space, as evidenced by zero-pretraining deep learning methods which are now achieving competitive performance with remarkably small networks (7M parameters). In parallel, four frontier AI labs (Anthropic, Google DeepMind, OpenAI, and xAI) reported ARC-AGI performance in public model cards in 2025, establishing ARC-AGI as an industry standard benchmark for AI reasoning. However, our analysis indicates that current frontier AI reasoning performance remains fundamentally constrained to knowledge coverage, giving rise to new forms of benchmark contamination. In this paper, we survey the top-performing methods, examine the role of refinement loops in AGI progress, discuss knowledge-dependent overfitting, and preview ARC-AGI-3, which introduces interactive reasoning challenges that require exploration, planning, memory, goal acquisition, and alignment capabilities.

1 Introduction: ARC-AGI

In 2019, François Chollet formalized a new definition of artificial general intelligence (AGI), characterizing it as a system capable of efficiently acquiring new skills and solving novel problems for which it was neither explicitly designed nor trained. (*7*) As a first concrete attempt to measure this new definition of intelligence, Chollet published the Abstraction and Reasoning Corpus (ARC) (*6*) (later renamed ARC-AGI to avoid naming collisions with other AI benchmarks.) The original benchmark dataset is referred to as ARC-AGI-1. It is a set of independent "tasks" (see figure 1), each consisting of a number of "demonstration pairs" (two or more, with a median count of three) and one or more "test inputs". A test pair consists of an "input grid", a rectangular grid of variable size (up to a maximum size of 30 rows by 30 columns) where each cell can have one of ten distinct "values", and an output grid which should be fully inferable from the characteristics of the input grid. The goal is to use the demonstration pairs to understand the nature of the task and use this understanding to construct the output grid corresponding to each test input. The test taker is allowed two attempts per test input.

The defining characteristic of the benchmark is that it should not be possible to prepare for any of the tasks in advance. Every task in the dataset follows a different underlying logic, requiring independent rule discovery. All tasks were created by humans to ensure a high degree of novelty and diversity.

ARC-AGI tasks do not require specialized world knowledge (e.g., historical facts) nor language to solve. The only assumed prior knowledge is Core Knowledge (*7*) – concepts such as objectness, basic topology, elementary integer arithmetic, etc. Human Core Knowledge has been investigated by Spelke et al. (*26*). These knowledge priors are acquired by children very early (typically before age four) and are universally shared by all humans. The ARC-AGI public training tasks are designed to expose test-takers to all the Core Knowledge priors needed to solve ARC-AGI tasks.

1.1 Dataset Composition

Following ARC Prize 2024, we released ARC-AGI-2 in early 2025. ARC-AGI-2 maintains the same task format and core principles as ARC-AGI-1 while incorporating more complex tasks with greater generalization difficulty, and normalizing the distribution of task difficulty.

ARC-AGI-2 consists of a larger set of human-created tasks split into three primary subsets:

- Public training tasks (400, imported from ARC-AGI-1) - Intended to demonstrate the task format and allow for learning the Core Knowledge priors.
- Semi-Private Evaluation tasks (120) - Used for intermediate leaderboard scoring during competitions. While not publicly released, these tasks have been exposed to commercial APIs and thus carry some risk of leakage.
- Private Evaluation tasks (120) - Used for final competition scoring and verification. This set is fully private and theoretically free from leakage.

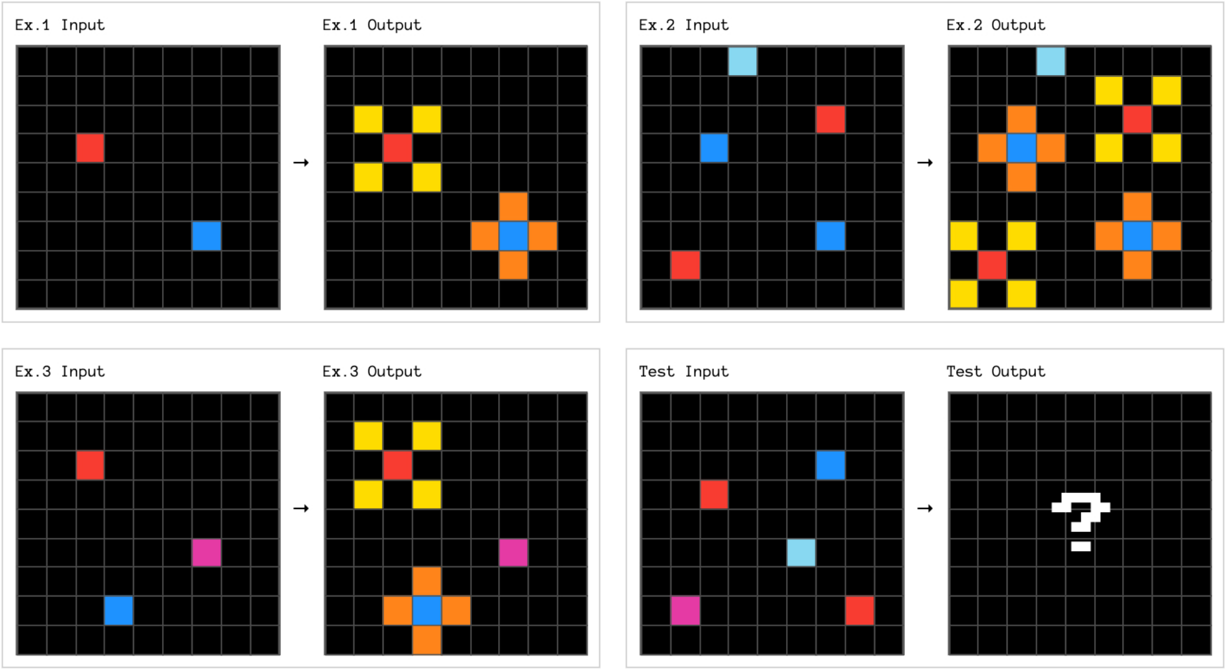


Figure 1: Example ARC-AGI task

State-of-the-art scores are only reported on the Semi-Private and Private Evaluation task sets to reduce the risk of overfitting and data contamination.

An important characteristic of ARC-AGI tasks is that they remain hard for AI systems, yet easy for humans. The original ARC-AGI-1 private evaluation tasks were tested by two people who scored 97% and 98%, and collectively solved 100% of tasks.

For ARC-AGI-2, after selection and validation, 100% of the tasks were solved by at least two (or more) independent, non-expert human testers drawn from the general public, with each task attempted by between 2 and 10 humans. The results of our human study establish that all tasks in ARC-AGI-2 are solvable by humans with no prior training.

2 ARC Prize 2025 Results

2.1 Competition Progress

ARC Prize 2025 ran from March 26, 2025, to November 3, 2025. In total, 1,455 teams submitted 15,154 entries to the Kaggle competition, at levels similar to ARC Prize 2024. The top competition score reached a new state-of-the-art on the ARC-AGI-2 private dataset of 24% at a compute cost of $0.20 per task.

The paper submission track saw significant growth, with 90 papers submitted, up from 47 in 2024. Due to the exceptional quality of submissions, we expanded the paper prizes to include 5 additional runners-up beyond the top three award winners, and recognized 8 additional honorable mentions.

All ARC Prize 2025 winning solutions and papers are open-source and available at arcprize.org.

2.2 Top Scores

Place	**Prize**	**Team**	**ARC-AGI-2 Private Score**
1st	$25k	NVARC (*25*)	24.03%
2nd	$10k	the ARChitects (*27*)	16.53%
3rd	$5k	MindsAI (*8*)	12.64%
4th	$5k	Lonnie (*24*)	6.67%
5th	$5k	G. Barbadillo (*4*)	6.53%

Table 1: ARC Prize 2025 Top Score winners.

The top three Kaggle submissions demonstrate continued progress in test-time training and ensemble techniques.

1st Place - NVARC (24.03%): This entry builds upon the 2024 ARChitects winning entry (which leverages test-time training) and makes heavy use of synthetic data generation to improve model performance.

2nd Place - the ARChitects (16.53%): A 2D-aware, masked-diffusion language model with recursive self-refinement and perspective-based scoring. This solution improved substantially over the team's 2024 autoregressive system through novel architectural modifications tailored for spatial reasoning.

3rd Place - MindsAI (12.64%): A heavily-engineered test-time-training pipeline that combines test-time fine-tuning, augmentation ensembles, tokenizer dropout, and novel pretraining techniques to produce a competitive score on ARC-AGI-2.

Video interviews with the 1st, 2nd, and 3rd place Top Score winners are available on the ARC Prize website (*3*).

2.3 Paper Awards

Place	Prize	Authors	Title
1st	$50k	A. Jolicoeur-Martineau	*Less is More: Recursive Reasoning with Tiny Networks* (*15*)
2nd	$20k	J. Pourcel, C. Colas, P. Oudeyer	*Self-Improving Language Models for Evolutionary Program Synthesis: A Case Study on ARC-AGI* (*23*)
3rd	$5k	I. Liao, A. Gu	*ARC-AGI Without Pretraining* (*19*)
Runner Up	$2.5k	I. Joffe, C. Eliasmith	*Vector Symbolic Algebras for the Abstraction and Reasoning Corpus* (*14*)
Runner Up	$2.5k	J. Berman	*From Parrots to Von Neumanns: How Evolutionary Test-Time Compute Achieved State-of-the-Art on ARC-AGI* (*5*)
Runner Up	$2.5k	E. Pang	*Efficient Evolutionary Program Synthesis* (*22*)
Runner Up	$2.5k	E. Guichard, F. Reimers, M. Kvalsund, M. Lepperod, S. Nichele	*ARC-NCA: Towards Developmental Solutions to the Abstraction and Reasoning Corpus* (*11*)

Table 2: ARC Prize 2025 Paper Award winners.

The top three Paper Awards recognized novel approaches that advance the theoretical and practical understanding of artificial fluid intelligence.

1st Place - Jolicoeur-Martineau: The Tiny Recursive Model (TRM) is a 7M-parameter, single-network recursive model with separate answer and latent states. Using deep supervised refinement, TRM demonstrates that extremely small networks can achieve competitive ARC-AGI performance when trained with appropriate recursive reasoning mechanisms.

2nd Place - Pourcel, Colas, and Oudeyer: SOAR (Self-improving Operators for Automated program Refinements) is a self-improving evolutionary program synthesis framework that fine-tunes an LLM on its own search traces. This approach improves open-source ARC-AGI-1 solution performance by up to 52% without requiring human-engineered domain-specific languages (DSLs) or solution datasets, demonstrating the potential for autonomous improvement in program synthesis systems.

3rd Place - Liao and Gu: CompressARC is an MDL-based (Minimum Description Length), single puzzle-trained neural code golf system that achieves 20-34% on ARC-AGI-1 and 4% on ARC-AGI-2 without any pretraining or external data. This work demonstrates that pure test-time optimization based on description length minimization can solve abstract reasoning tasks without leveraging large-scale pretraining.

Video interviews with the Paper Award winners are available on the ARC Prize website channel (*3*).

2.4 Honorable Mentions

Eight additional papers received honorable mention recognition for their contributions to ARC-AGI research:

- K. Hu et al., *"ARC-AGI is a Vision Problem!"* (*13*)
- D. Franzen, J. Disselhoff, D. Hartmann, *"Product of Experts with LLMs: Boosting Performance on ARC Is a Matter of Perspective"* (*10*)
- G. Barbadillo, *"Exploring the combination of search and learn for the ARC25 challenge"* (*4*)

- A. Das, O. Ghugarkar, V. Bhat, J. McAuley, *"Beyond Brute Force: A Neuro-Symbolic Architecture for Compositional Reasoning in ARC-AGI-2"* (*9*)
- R. McGovern, *"Test-time Adaptation of Tiny Recursive Models"* (*20*)
- P. Acuaviva et al., *"Rethinking Visual Intelligence: Insights from Video Pretraining"* (*1*)
- J. Cole, M. Osman, *"Don't throw the baby out with the bathwater: How and why deep learning for ARC"* (*8*)
- I. Sorokin, J. Puget, *"NVARC solution to ARC-AGI-2 2025"* (*25*)

3 Program Refinement Loops

The central theme driving AGI progress in 2025 is the emergence of the refinement loop. At its core, a refinement loop iteratively transforms one program or model version into a slightly better one, based on a feedback signal.

3.1 Types of refinement loops

Representatives of refinement loops include:

- **Deep learning with test-time training methods**, where the program being refined is the weights of a pretrained model.
- **Zero-pretraining deep learning methods** such as TRM.
- **Evolutionary Program Synthesis** in either symbolic program space or natural language program space.
- **Test-time Chain-of-Thought optimization** with feedback from a verifier model.

Among these, two represent especially interesting developments for 2025: Evolutionary Program Synthesis and Zero-pretraining deep learning methods.

3.1.1 Evolutionary Program Synthesis

Examples of this technique include J. Berman (*5*) and E. Pang (*22*). Berman's approach employs an evolutionary search harness that evolves ARC solution programs in natural language. Pang's approach follows a similar strategy but operates in Python and dynamically creates a program abstraction library to guide synthesis.

Both approaches implement a two-phase refinement process. First, an exploration phase generates many candidate solutions. Second, a verification phase analyzes these programs to produce a feedback signal. This cycle repeats per task until the resulting program is fully refined and provides accurate answers for all training input/output pairs.

3.1.2 Zero-Pretraining Deep Learning Methods

Refinement loops are also becoming the basis for a new type of training paradigm for deep learning models.

Classically, deep learning models are trained on input/output pairs using gradient descent to create a static neural network. This training algorithm gradually refines a high-dimensional curve in the network's latent space. At inference time, when presented with a new input, the network performs forward passes to approximate the output based on this curve. In 2023 and 2024, this paradigm was expanded to add test-time training, where the network is further fine-tuned at inference time on examples of a novel task (a process which is itself a kind of program refinement loop). In this paradigm, the neural weights are trained to represent a task-specific solver program. Test-time training is responsible for the top scores in both ARC Prize 2024 (the ARChitects) and 2025 (NVARC).

Input/output pairs are still used as ground truth and refinement loops play a key role in training, mirroring program synthesis approaches but operating in weight space rather than symbolic space.

More recent work demonstrates early success with a version of this that does away with pretraining entirely, initializing a network from scratch for a specific task and fitting the weights to encode a task-solving program using only examples of that task. Examples include Liao et al. (*19*), Hierarchical Reasoning Models (HRM) (*28*) and Jolicoeur-Martineau (*15*).

This approach exhibits two unusual properties:

1. The resulting networks are extremely small relative to their ARC-AGI performance.
2. All material task-specific training occurs at test time.

3.2 Open Source Examples

3.2.1 Tiny Recursive Model (TRM)

The Tiny Recursive Model (TRM) (Paper Award 1st Place, Jolicoeur-Martineau (*15*)), which builds upon the earlier Hierarchical Reasoning Model (HRM) (*28*), achieves 45% test accuracy on ARC-AGI-1 and 8% on ARC-AGI-2 with only a 7M parameter network. From the paper:

> Tiny Recursive Model (TRM) recursively improves its predicted answer y with a tiny network. It starts with the embedded input question x and initial embedded answer y, and latent z. For up to $N_{sup} = 16$ improvement steps, it tries to improve its answer y. It does so by i) recursively updating n times its latent z given the question x, current answer y, and current latent z (recursive reasoning), and then ii) updating its answer y given the current answer y and current latent z. This recursive process allows the model to progressively improve its answer (potentially addressing any errors from its previous answer) in an extremely parameter-efficient manner while minimizing overfitting.

3.2.2 CompressARC

CompressARC (Paper Award 3rd Place, Liao and Gu (*19*)) uses only 76K parameters, yet achieves 20% on the ARC-AGI-1 evaluation set, processing each puzzle in approximately 20 minutes on a single NVIDIA RTX 4070 GPU.

This solution features three distinctive characteristics:

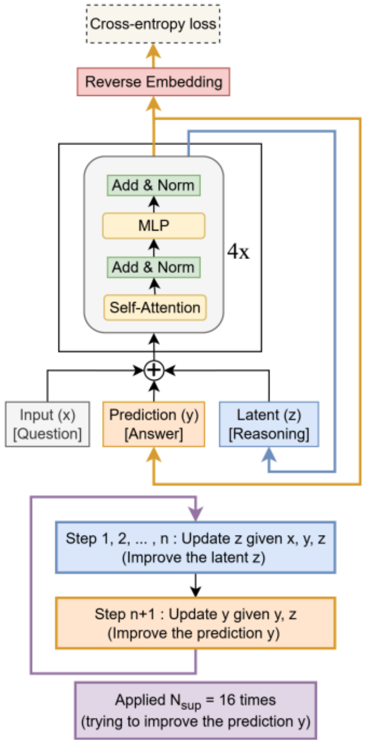

Cross-entropy loss
Reverse Embedding
Add & Norm
MLP
Add & Norm
Self-Attention
4x
Input (x) [Question]
Prediction (y) [Answer]
Latent (z) [Reasoning]
Step 1, 2, ... , n : Update z given x, y, z (Improve the latent z)
Step n+1 : Update y given y, z (Improve the prediction y)
Applied N_{sup} = 16 times (trying to improve the prediction y)

Figure 2: TRM architecture.

- **No pretraining:** Models are randomly initialized and trained only at test time.
- **No dataset:** One model trains on a single target task and produces one answer.
- **No branching search:** The approach relies solely on gradient descent.

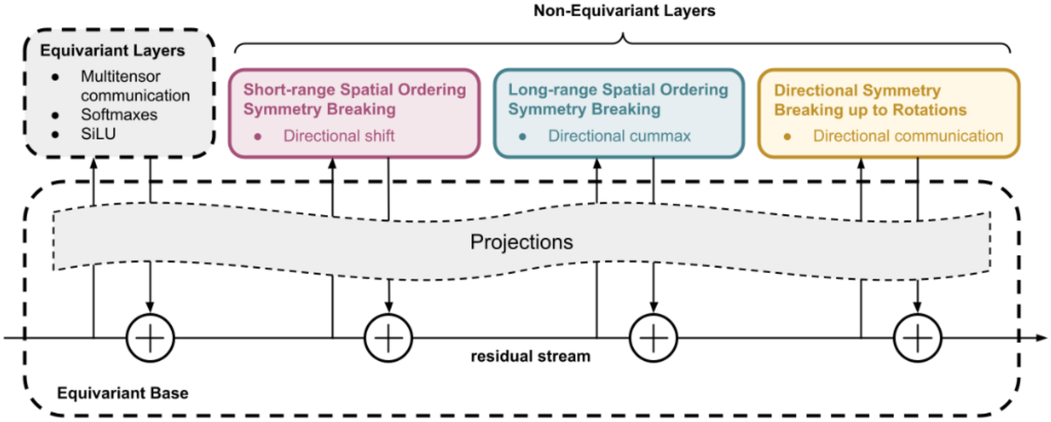
The method operates by minimizing the description length of each task at test time, following the Minimum Description Length (MDL) principle. Liao derives how a standard Variational Autoencoder (VAE) loss with decoder regularization can substitute for combinatorial search to refine very small neural network programs. The generalization achieved by such a compact network is remarkable.

3.3 Commercial Examples

Evidence of iterative refinement appears in commercial AI reasoning systems. A Chain-of-Thought can be interpreted as a natural language program that transforms one latent state into another.

Consider ARC-AGI-1 task #**4cd1b7b2**. Gemini 3 Pro used 96 reasoning tokens to solve this task, whereas Gemini 3 Deep Think employed 138,000 tokens. Higher reasoning modes for these systems exhibit a strong correlation with increased reasoning token counts (longer programs), even when not strictly necessary for task completion.

These extended natural language programs enable more refinement through additional exploration and verification cycles. Analysis of reasoning outputs from commercial systems reveals self-corrective behavior:


Non-Equivariant Layers
Equivariant Layers
Multitensor communication
Softmaxes
SiLU
Short-range Spatial Ordering Symmetry Breaking
Directional shift
Long-range Spatial Ordering Symmetry Breaking
Directional cummax
Directional Symmetry Breaking up to Rotations
Directional communication
Projections
residual stream
Equivariant Base

Note: Decoding of z, linear heads, and layernorms are not shown. They are included in the equivariant base.

Figure 3: CompressARC architecture and approach.

> *... which fails the complete set requirement. This suggests the current solution might not fully satisfy the puzzle's constraints. I need to re-examine the box configuration and explore alternative arrangements ...* (Claude Opus 4.5)

> *... which suggests further investigation is needed to complete the analysis. I'll verify the center point at row 9, column 15 ...* (Claude Opus 4.5)

> *... maybe each input row is being duplicated three times in the output, but how does that fit with the rest? Wait, the third output row is ...* (QwQ 32B)

An important finding with frontier commercial models released in late 2025 (Gemini 3, Claude Opus 4.5, etc.) is that refinement loops can be implemented at the application layer to meaningfully improve task reliability, rather than relying solely on provider reasoning systems. This approach still requires that the foundational model possess adequate knowledge coverage of the task domain.

3.4 Model Refinement Harnesses

In the last quarter of 2025, we introduced a new leaderboard category termed "Model Refinements" to track application-layer improvements to commercial AI systems, also known as harnesses. We verified a Gemini 3 Pro refinement harness implementation open-sourced by Poetiq, which improves performance on ARC-AGI-2 from a baseline of 31% accuracy at $0.81 per task to 54% accuracy at $31 per task. The same refinement approach achieved comparable gains on Claude Opus 4.5, with accuracy rivaling Gemini 3 Pro but at about twice the cost per task (approximately $60 per task), as reported by Poetiq.

Currently, the refinement harnesses we observe are domain-specific. However, techniques such as GEPA (*18*) and DSPy (*17*) enable the development of general-purpose reliability improvements at the application layer, provided a verifier or environment capable of producing a feedback signal is available.

We anticipate that general refinement harness improvements will eventually be integrated behind the API of commercial AI systems. Simultaneously, we expect that bleeding-edge, task-specific accuracy will continue to be driven by knowledge specialization and application-layer verifiers.

4 AGI Progress & The Future of ARC-AGI

As of 2025, with the advent of AI reasoning systems, task domains with the following two characteristics are reliably automatable, with no new science needed:

1. Sufficient task knowledge coverage exists in the pretraining corpus.
2. The task provides a verifiable feedback signal.

Current AI reasoning performance is fundamentally related to model knowledge. This relationship warrants careful consideration, as human reasoning capability is not similarly bound to knowledge. This coupling has various implications and leads to imprecise characterizations such as "jagged intelligence." (*16*)

Supporting evidence for this knowledge-dependent reasoning emerged across multiple domains in 2025, including performance on ARC-AGI-2 (abstract reasoning), 2025 IMO Gold Medal achievement (mathematics), and 2025 ICPC 100% performance (competitive programming) – all driven by AI reasoning systems. These task domains are substantially broader than those addressable by pure language models without reasoning capabilities. However, they remain relatively narrow in a global context.

The invention and scaling of chain-of-thought synthesis represents a profound upgrade in AI capability comparable to the invention and scaling of transformers. However, we are still in the early stages of deployment. Few users have directly experienced these tools. According to Sam Altman (OpenAI), approximately 7% of ChatGPT free users have engaged with "thinking" mode (*2*). We expect diffusion of current technology to require 5-10 additional years, even within business contexts alone.

Collecting domain knowledge and building verifiers is not cost-free. This represents relatively expensive and specialized work. Presently, AI automation is a function of the societal willingness to invest in the necessary talent, compute, and data resources. We anticipate steady progress over the next 12-24 months as society conducts a global search for problems that are both (1) most important and (2) fall within acceptable cost thresholds. This includes early results in which AI systems produce novel scientific knowledge in fields with adequate knowledge coverage. Recent work by Hsu reports an AI refinement loop using a generator–verifier architecture to produce novel results in quantum physics (*12*).

However, many potentially automatable problems fall beyond current societal cost cutoffs. As engineering advances, costs will decrease, expanding the set of domains that can be automated. More broadly, machines capable of highly efficient adaptation to produce paradigm-shifting innovation remain firmly within the realm of science fiction.

For the ARC-AGI-1 and ARC-AGI-2 format, we assess that the Grand Prize accuracy gap is now primarily bottlenecked by engineering, while the efficiency gap remains bottlenecked by fundamental science and new ideas. ARC Prize exists to inspire and reward open AGI progress, and as previously committed, we will continue operating the ARC-AGI-2 Grand Prize competition in 2026 to track progress toward a fully open and reproducible solution.

Despite their capabilities, AI reasoning systems still exhibit numerous flaws and inefficiencies necessary to overcome in order to reach AGI. We still need new ideas, such as methods to separate knowledge and reasoning, among other challenges. New benchmarks will be needed to highlight the moment in which those ideas arrive.

4.1 Knowledge Overfitting

In machine learning, overfitting occurs when a model learns excessive detail from training data. The model memorizes exact training examples rather than learning general patterns, leading to poor performance on unseen test data. A common AI benchmarking critique is that model providers are incentivized to "benchmark maximize" or "train to the test" to report high scores for marketing purposes that do not generalize to real-world applications. ARC-AGI-1 and ARC-AGI-2 were designed to resist this style of overfitting by employing a private dataset for official scoring and verification.

AI reasoning systems have altered the landscape in a manner that reflects genuine progress. They have demonstrated non-zero fluid intelligence and can adapt to tasks somewhat removed from their precise knowledge base when the foundational model is grounded in the broader domain. This means that even well-designed benchmarks resistant to direct memorization can now be "overfit" if the public training and private test sets are too similar (e.g., independent and identically distributed) and the model has been trained on substantial public domain data.

We assert that this phenomenon is now occurring with ARC-AGI-1 and ARC-AGI-2 – accidentally or intentionally, although we cannot determine which.

Evidence from our Gemini 3 verification demonstrates this pattern:

> *... Target is Green (3). Pattern is Magenta (6) Solid. Result: Magenta Square on Green ...*
> (Gemini 3 Deep Think)

Our LLM verification harness does not mention ARC-AGI tasks or color formats, yet the model employs correct ARC color mappings in its reasoning. This strongly suggests that ARC data are well-represented in the underlying model – sufficiently so to make correct ARC inferences based solely on the structure and format of 2D JSON arrays of integers.

4.2 Characterizing AGI through continual benchmark adaptation

Although we assess that this new form of "overfitting" assists models in solving ARC, we cannot precisely quantify the magnitude of this effect. Regardless, the ARC-AGI-1 and ARC-AGI-2 formats have provided a valuable scientific indicator of AI reasoning progress. However, benchmark design must adapt.

In fact, ARC Prize has revealed a broader lesson over the past two years: the most valuable and effective benchmarks are created by teams fundamentally committed to driving progress. Such progress requires a sustained dedication to understanding the underlying technology through serious study, a willingness to identify flaws and incentivize corrective action, and adaptation as the technology improves. It also requires a year-over-year commitment. Building effective benchmarks demands sustained effort.

The critical concept is *adaptation*. Adaptation represents the core mode of intelligence. This process extends beyond creating effective benchmarks – it constitutes the ultimate measure of general intelligence itself.

From François Chollet in December 2024 (*21*):

> *You'll know AGI is here when the exercise of creating tasks that are easy for regular humans but hard for AI becomes simply impossible.*

This captures the ARC-AGI benchmark design methodology: operate a real world refinement loop by iteratively improving benchmarks in response to AI progress to drive the gap between "easy for humans, hard for AI" toward zero.

By this definition, we have not yet achieved AGI. We are actively developing ARC-AGI-3 for release in early 2026 and are optimistic about the new format. We anticipate that it will stimulate the development of entirely novel ideas.

4.3 ARC-AGI-3

Over the past six months, we have focused on developing ARC-AGI-3. Like all versions of ARC, it is designed to be "easy for humans, hard for AI," while serving as the most valuable and scientifically useful benchmark pointing toward AGI and identifying what remains necessary to achieve it. We are building hundreds of never-before-seen interactive environments designed to test agentic reasoning.

We plan to release ARC-AGI-3 in early 2026 alongside ARC Prize 2026. This new benchmark version marks the first major format change since ARC-AGI was introduced in 2019. While the first two versions challenged static reasoning, the third version is designed to challenge interactive reasoning and requires new AI capabilities to succeed: exploration, planning, memory, goal acquisition, and alignment.

Efficiency is a fundamental aspect in the measurement of intelligence. We are particularly optimistic that the ARC-AGI-3 scoring metric will enable formal comparison of human and AI action efficiency (i.e., learning efficiency) for the first time.

5 Conclusions

ARC Prize 2025 demonstrated continued open-source progress towards AGI, with the top Kaggle score reaching 24% on ARC-AGI-2 and significant growth in paper submissions (90, up from 47 in 2024). The emergence of the refinement loop as a central theme represents a significant shift in approaches to abstract reasoning, evidenced by both zero-pretraining deep learning methods and commercial AI reasoning systems.

The year revealed that AI reasoning performance remains fundamentally constrained by knowledge coverage, a characteristic distinct from human reasoning which is capable of extreme generalization. While this enables reliable automation of tasks with sufficient knowledge coverage and verifiable feedback signals, it also introduces new forms of overfitting that require adaptive responses in benchmark design.

ARC-AGI has served as a valuable scientific indicator for AI reasoning progress, with four major AI labs (Anthropic, Google DeepMind, OpenAI, and xAI) reporting ARC-AGI performance on model cards. However, the benchmark must continue to evolve. The development of ARC-AGI-3, scheduled for release in early 2026, represents this commitment to adaptation – the core principle of intelligence itself.

We remain committed to operating ARC Prize annually until the benchmark is defeated with a public reference solution. All ARC Prize 2025 winning solutions and papers are open-source, reproducible, and available at arcprize.org. We aim for ARC-AGI to continue serving as a focal point for research on generalization and reasoning, and to support sustained open progress towards AGI.

6 Appendix

6.1 Acknowledgments

ARC Prize 2025 builds on the foundation established by previous ARC-AGI competitions, including the 2020 Kaggle competition and the 2022-2023 ARCathons. We are grateful for the continued evolution of the benchmark and the growing community it has fostered.

ARC Prize would not be possible without the full support of the ARC Prize team, our competition partners at Kaggle, and our sponsors. Kaggle plays a critical role in the artificial intelligence and machine learning ecosystem, and their continued partnership has been instrumental in the success of ARC Prize 2025.

We extend our gratitude to all frontier AI labs who worked with us in 2025 to verify their new AI systems on ARC-AGI, including OpenAI, xAI, Anthropic, and Google DeepMind. This collaboration has been invaluable in establishing ARC-AGI as a meaningful benchmark for reasoning progress.

We recognize the dedication of our community members who have built tools, answered questions, and served as resources for researchers and participants throughout the year.

Finally, we extend our deepest gratitude to all participants in ARC Prize 2025, especially those who shared their work openly with the community. Your dedication advances the broader field of AI, bringing us closer to realizing the transformative potential of AGI for humanity. We are inspired by everyone with new ideas who works on ARC-AGI and remain committed to stewarding this attention as a north star toward AGI.

References

1. Paolo Acuaviva et al. Rethinking Visual Intelligence: Insights from Video Pretraining. `https://arxiv.org/abs/2510.24448`, 2025.
2. Sam Altman. Post on ChatGPT “thinking” mode adoption. `https://x.com/sama/status/1954603417252532479`, 2025. Accessed: 2026-01-13.
3. ARC Prize. ARC Prize 2025 Winner Interviews. `https://arcprize.org/competitions/2025/#:~:text=Jean%2DFran%C3%A7ois%20Puget-,WINNER%20INTERVIEWS,-TOP%20SCORES`, 2025. Featured interviews with ARC Prize 2025 winners.
4. Guillermo Barbadillo. Exploring the Combination of Search and Learn for the ARC25 Challenge. `https://ironbar.github.io/arc25/05_Solution_Summary/`, 2025.
5. Jeremy Berman. From Parrots to Von Neumanns: How Evolutionary Test-Time Compute Achieved State-of-the-Art on ARC-AGI. `https://github.com/jerber/arc-lang-public/blob/main/from_parrots_to_von_neumanns.pdf`, 2025.
6. François Chollet. Abstraction and Reasoning Corpus for Artificial General Intelligence (ARC-AGI). `https://github.com/fchollet/ARC-AGI`, 2019.
7. François Chollet. On the Measure of Intelligence. `https://arxiv.org/abs/1911.01547`, 2019.
8. Jack Cole and Mohammad Osman. Don’t throw the baby out with the bathwater: How and why deep learning for ARC. `https://arxiv.org/abs/2506.14276`, 2025.
9. Ashish Das, Omkar Ghugarkar, Vedant Bhat, and Julian McAuley. Beyond Brute Force: A Neuro-Symbolic Architecture for Compositional Reasoning in ARC-AGI-2. `https://github.com/CoreThink-AI/Research-publications/blob/main/Preprints/Beyond_Brute_Force__A_Neuro_Symbolic_Architecture_for_Compositional_Reasoning_in_ARC_AGI_2.pdf`, 2025.
10. Daniel Franzen, Jan Disselhoff, and David Hartmann. Product of Experts with LLMs: Boosting Performance on ARC Is a Matter of Perspective. `https://drive.google.com/file/d/1o1gGmlQTo6tsXzQ1T6NqTOqrP8hXNXnV/view`, 2025.
11. E. Guichard, F. Reimers, M. Kvalsund, M. Lepperod, and S. Nichele. ARC-NCA: Towards Developmental Solutions to the Abstraction and Reasoning Corpus. `https://etimush.github.io/ARC_NCA/`, 2025.
12. Steve Hsu. Post on AI generator–verifier refinement producing novel results in quantum physics. `https://x.com/hsu_steve/status/1996034522308026435`, 2025. Accessed: 2026-01-13.
13. Keya Hu et al. ARC-AGI is a Vision Problem! `https://arxiv.org/abs/2511.14761`, 2025.
14. Ian Joffe and Chris Eliasmith. Vector Symbolic Algebras for the Abstraction and Reasoning Corpus. `https://github.com/ijoffe/ARC-VSA-2025/blob/main/paper/paper.pdf`, 2025.

15. Alexia Jolicoeur-Martineau. Less is More: Recursive Reasoning with Tiny Networks. `https://arxiv.org/abs/2510.04871`, 2025.

16. Andrej Karpathy. Jagged Intelligence. `https://x.com/karpathy/status/1816531576228053133`, 2024. Social media post describing uneven performance of AI models.

17. Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. Dspy: Compiling declarative language model calls into self-improving pipelines, 2024.

18. Lakshya A Agrawal. GEPA: System Optimization through Reflective Text Evolution. `https://github.com/gepa-ai/gepa`, 2025. Open-source framework for optimizing arbitrary systems composed of text components, like AI prompts, against any evaluation metric.

19. Isaac Liao and Andrew Gu. ARC-AGI Without Pretraining. `https://iliao2345.github.io/blog_posts/arc_agi_without_pretraining/ARC_AGI_Without_Pretraining.pdf`, 2025.

20. Ronan McGovern. Test-time Adaptation of Tiny Recursive Models. `https://trelis.com/wp-content/uploads/2025/11/mcgovern_test_time_adaptation_trm.pdf`, 2025.

21. François Chollet Mike Knoop. OAI o3 Public Breakthrough. `https://arcprize.org/blog/oai-o3-pub-breakthrough`, December 2024. ARC Prize blog post containing a statement on the definition and recognition of AGI. Accessed: 2026-01-13.

22. E. Pang. Efficient Evolutionary Program Synthesis. `https://open.substack.com/pub/ctpang/p/arc-agi-2-sota-efficient-evolutionary`, 2025.

23. Julien Pourcel, Clement Colas, and Pierre Oudeyer. Self-Improving Language Models for Evolutionary Program Synthesis: A Case Study on ARC-AGI. `https://openreview.net/pdf?id=z4IG090qt2`, 2025.

24. Lonnie Qin. ARC Prize 2025 Competition Writeup (5th Place). `https://www.kaggle.com/competitions/arc-prize-2025/writeups/arc-prize-2025-competition-writeup-5th-place`, 2025.

25. I. Sorokin and Jean-Francois Puget. NVARC Solution to ARC-AGI-2 2025. `https://drive.google.com/file/d/1vkEluaaJTzaZiJL69TkZovJUkPSDH5Xc/view`, 2025.

26. Elizabeth S. Spelke and Katherine D. Kinzler. Core knowledge. *Developmental science*, pages 89–96, 2007.

27. the ARChitects. ARC2025 Solution by the ARChitects. `https://lambdalabsml.github.io/ARC2025_Solution_by_the_ARChitects/`, 2025.

28. Guan Wang et al. Hierarchical reasoning model. *arXiv preprint arXiv:2506.21734*, 2025.