

超越模式识别的神经网络——一种符号主义和联结主义结合的新范式

前言部分

- **标题:** 超越模式识别的神经网络——一种符号主义和联结主义结合的新范式
- **作者:** Ball Lightning
- **摘要:** 本研究挑战了神经网络作为统计模式识别器的传统观念，提出并验证了一种能够唤醒其内生精确推理能力的新范式。该范式通过采用程序化生成的理想数据与并行的、非自回归的求解框架，将标准神经网络从概率性的模仿者转变为确定性的规则执行器。我通过在符号规则，算法拟合，图像推理，物理模拟，可解释性等一系列任务上的成功，有力的证明了这个范式的通用性。我又在一维元胞自动机的多步演化任务上，对多种代表性架构（MLP、CNN、RNN、UNET、Diffusion）进行了系统性评估，并证实了非transformer的多种模型也具有类似能力。这一发现强有力地证明，精确的算法执行能力是联结主义系统的固有潜能，而非特定架构的专利。本工作通过**神经雕刻范式**，首次系统性地证明了标准神经网络具备精确执行任意算法的内在潜能，为构建高可靠、可解释的通用人工智能系统开辟了一条全新的道路。**代码已开源于：**<https://github.com/ball-lightning6/neural-sculpting-paradigm>。

正文部分 (Main Body)

第一幕：现代AI的推理鸿沟

深度神经网络在模式识别任务上取得了巨大成功，然而，在需要精确、多步逻辑推理的领域，其表现出的“幻觉”和逻辑不一致性，始终是一个根本性的瓶颈。这限制了AI在科学、数学等高可靠性领域的应用。本文认为，这一困境的根源，并非神经网络的内在缺陷，而是其架构的归纳偏置与任务的内在结构之间的根本性错配。

基于此核心洞察，我提出并验证了一套全新的、简洁而强大的求解式推理范式——我称之为**神经雕刻范式**。该范式通过放弃自回归，采用单步结构化求解输出，以及使用程序化生成的理想数据，系统性地将一个标准的 [2] Transformer，从一个统计模仿者，转变为一个精确规则执行器。我在横跨符号规则、算法学习、图像推理、物理模拟和可解释性等一系列任务上的广泛实验，不仅验证了该范式的普遍有效性，更通过语义洗牌等实验，证明了模型学到的是抽象的代数结构，而非表面的符号模式。我的工作，为构建可信、可解释、具备高阶推理能力的AI系统，提供了一条简洁、通用且可扩展的新路径。

第二幕：神经符号计算简史

将神经网络强大的学习能力与符号系统严谨的推理能力相结合的“神经符号计算”，一直是人工智能领域孜孜以求的**圣杯**。前人的探索，主要沿着两条路径展开。

第一条路径，是“架构的改造”。这类工作，试图通过设计特殊的、模仿符号计算过程的网络架构，来赋予模型推理能力。其开创性的探索，可以追溯到引入外部记忆模块的 [6] 神经图灵

机，以及其后续发展出的 [7] 可微神经计算机。这一思想，在后来的 [11] 神经模块网络和 [10] 图神经网络中，得到了进一步的延续。这些工作的共同特点，是试图通过“为推理而设计专门的硬件”，来增强神经网络的能力。然而，这些特化的架构，往往难以迁移到新的任务领域，并面临着设计复杂的困境。

第二条路径，是“逻辑的嵌入”。这类工作，试图将外部的符号逻辑规则，以一种可微分的形式，“嵌入”到神经网络的学习过程中。无论是通过模糊逻辑将规则转化为损失函数约束的 [8] 神经定理证明器，还是将神经网络作为概率逻辑程序一部分的 [9] DeepProbLog，其核心，都是试图用一个“外部的”逻辑系统，来“指导”或“约束”一个“内部的”神经网络。

然而，上述路径，都共享着一个共同的、底层的“混合式”哲学，即将“学习”与“推理”视为需要被外部“粘合”的两种异构能力。

与此根本不同，我的工作发现，推理，是Transformer架构的一种“原生的、沉睡的”能力，它不需要“混合”，而需要被“唤醒”。我发现，通过一个全新的、简洁的训练范式——即，“求解式输出”与“理想数据”的结合——一个标准的、通用的Transformer架构，就能够自发地、系统性地，学会并执行精确的符号规则。我的贡献，不在于设计一个更复杂的“模型”，而在于发现了一套更正确的“方法”，来解锁早已存在于模型之中的巨大潜力。

第三幕：神经雕刻范式

我发现的新范式，其核心思想并非设计一种全新的模型架构，而是通过对学习环境与任务定义的根本性重构，系统性地证明了神经网络内在强大的推理能力，这种推理能力的问题形式不限于符号规则、算法问题、图像推理、物理模拟、可解释性。如果要用一句话概括，那就是一类有明确变换关系的问题，即对应相应的输入，有一个机械可执行的符号化过程，变换为唯一的精确的输出。

我的主要实验全部基于标准Transformer架构以及它在图像领域的变体比如 [3] ViT或者 [4] Swin Transformer。但是我也做了少部分实验证明mlp, rnn, cnn等等网络也有类似的能力，尽管可能并不如transformer强大。这证明这个能力并非transformer独有，而是神经网络的内在固有能力。

该范式建立在以下三个相互协同的核心原则之上。

3.1. 核心引擎：以“抽象关系”为偏置的Transformer

尽管其他类型的神经网络也有此类能力，但实验证明transformer最适合做此类推理问题，它的此类能力是最强的。原因可能是Transformer架构的核心自注意力机制从根本上打破了空间的束缚。它允许序列中的任何一个元素，直接地、并行地，与其他所有元素进行交互，并计算它们之间的“关系强度”。这种架构，天然地，就与“推理”这个任务的内在结构，存在着深刻的“同构性”。它不是在“看”像素的邻域，它是在“理解”整个符号系统的“关系图谱”。实验发现不管是transformer, ViT, Swin Transformer等等都具有同样的能力，即前面所列出的符号规则、算法问题、图像推理、物理模拟、可解释性等等。因此我可以认为这些问题的分类只是表象，它们之间可能并没有本质区别。

3.2. 输出范式：从“序列模仿”到“并行求解”

当前大型模型推理能力主要采用自回归预测下一个token的方式，而我采用完全抛弃自回归的一次预测方式。在此类推理问题上，这个方法更成功的原因可以解释如下，抛弃自回归的方法，输入数据和输出数据的格式更加纯粹和一致，而自回归预测下一个token的方式，天然就无法做到这一点，它可能还是更适合做大语言模型。而且这种“逐词预测”的模式，存在一个根本性缺陷：错误累积，序列中任何一个早期的微小错误，都可能在后续的生成中被不断放大，导致逻辑链的彻底崩溃。而我的这种“一步到位”的格式，迫使模型在一次前向传播中，就必须对其内部的所有“神经元”进行全局性的协同计算和自治性约束，从而将推理过程，从一个脆弱的“线性猜测链”，转变为一个鲁棒的“并行约束求解”过程。

3.3. 数据范式：从“现实噪声”到“理想法则”

传统机器学习范式，往往因为数据的真实性，被迫从充满噪声、偶然相关性和不确定性的数据中，清洗出知识。这种数据往往在输入和输出端充满噪声，而且并没有一个明确的精确的变换规则。因此，在这种训练数据中，神经网络只可能体现出模式识别的能力。然而，对于学习精确的、确定性的规则而言，这种“脏数据”，本身就是一种强大的“干扰源”。

我的范式，则采取了一种截然相反的“数据哲学”。我通过程序化脚本，来生成海量的、逻辑上绝对纯粹的“理想数据”。在这些数据中，输入和输出之间，只存在唯一的、必然的、精确的逻辑关系，而不存在任何统计上的“捷径”或“偶然模式”。这种“信噪比”极高的学习环境，极大地降低了学习的难度，它让模型别无选择，只能去学习那个隐藏在所有样本背后，唯一的、不变的“信号”——即，底层的规则本身。我大量的实验（详见第四章）证明，在这种范式下，模型只需要覆盖总输入空间中一个可以忽略不计的微小部分，就能实现对整个问题空间的完美泛化。

另外，训练数据必须和输入空间的分布相同（或者说目前看来这样做最方便），这个条件我在训练集生成脚本中用随机采样达成。这个范式也无法泛化到输入空间之外，或者换句话说，由于训练数据占整个输入空间的比例非常小，除掉训练数据的整个剩下的极大的输入空间，收敛后的模型都可以精确泛化，这个可以类比过去范式的泛化。

3.4. 范式解读：用梯度下降，去雕刻精确规则

综上所述，我的范式，可以被理解为一种用联结主义的过程，去无限逼近符号主义的理想哲学实践。这就像一位雕塑家，面对一块粗糙的大理石（一个随机初始化的神经网络）。“理想数据”为他提供了关于“大卫像”的、无数个角度的、完美的“照片”；“求解式输出”的损失函数，则告诉他，他当前的作品，与那个“理想”之间，还存在怎样的“差距”；而“梯度下降”，就是他手中那把，每一次，都能朝着正确方向，凿去一小块多余石料的、神奇的“刻刀”。

通过数百万次的、微小而精确的“雕琢”，最终，那座代表着“精确规则”的、完美的“大卫像”，就自然地，从那块“连续的、混沌的”神经网络石料中，“涌现”了出来。而在某些无法完美收敛的问题上，这个范式也总是可以压缩输出的不确定性。

其实我发现的范式，简单来说就是用联结主义的梯度下降，去拟合一个精确的规则，这是联结主义和符号主义连接在一起的极其自然的方式。我总结这个范式，其实就是多模态，可以端到端学习几乎一切有精确定义的变换规则。

3.5. 模型描述

符号推理/算法推理——qwen2_0.5b [21] 微调lora+自定义lm_head 或者后面换成全量训练一个小transformer

图像推理符号——ViT/Swin Transformer

多模态/image2image——Swin Transformer+unet

文字推理生成图像——一个小transformer后接unet

对应代码：

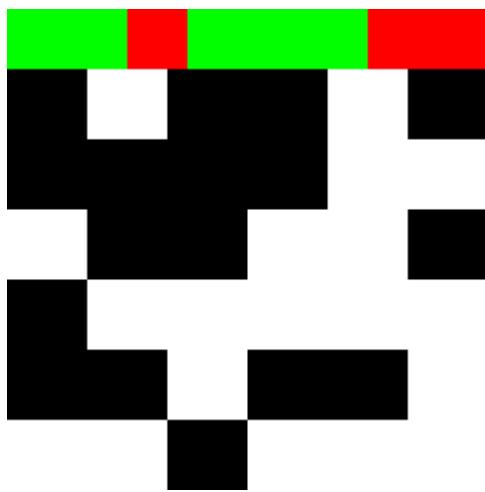
多模态/image2image：train_image2image.py

符号推理/算法推理：train_tiny_transformer.py

文字推理生成图像：train_text2image.py / train_qwen2_text2image.py

图像推理符号：train_swin_image2text.py

关于多模态推理，我暂时使用图像输入，把符号输入编码进入图像的方式来解决，即Swin Transformer+unet，未来应该有其他方法。例子如下，最上层8位绿色和红色方块，代表元胞自动机的规则，下面6*6黑白色格子，代表1位36位元胞自动机的初始状态，这种方法在预测应用两层元胞自动机规则后的状态的任务中取得很好效果。



另外，mlp, cnn, rnn也可以完成简单任务，其他模型也有潜力完成这个类型的任务。但我仍然以transformer为标准模型。

第四幕：跨越多领域的实证

4.1. 基础能力：符号规则学习

本节旨在从最根本的层面，验证本范式学习精确、复杂符号规则的核心能力。

训练集验证集划分比例：一般按照1000step左右的频率进行一次eval，以便随时观察eval loss下降趋势，按照这个频率计算出不至于让eval时间过长的验证集比例，验证集占数据集比例一般在0.01以下，由于验证集和数据集和输入空间同分布，可以验证输出可靠的eval loss。

关于模型训练超参数：几乎所有超参数都是不需要改变的，除了有时候提高显存使用会在一些比较简单的任务上提高batchsize，以及一些涉及图像推理的任务训练有时候loss会变为nan，这样情况会减半学习率。

模型结构变化：输出符号相关的一般最重要地是需要改变多标签二分类的标签个数。

输入空间估计：有些任务的输入空间难以直接计算，我采用类似蒙特卡洛法的方法估计，即产

生任务相关的随机数（这些随机数足以确定一个样本）一个比较大的次数，根据重复样本数估算输入空间的量级。

4.1.1. 元胞自动机

任务描述：模型学习预测和输出一个一维元胞自动机给定规则变换给定次数后的状态

输入格式：30位的由“0”或者“1”组成的字符串，表示元胞自动机的初始状态

输出格式：30个标签的多标签二分类格式，等同于30个01字符串，表示元胞自动机变换后的状态

loss：平均二元交叉熵损失

训练配置：4090显卡

训练集生成脚本：generate_cellular_automata_1d.py

数据集：ca_rule110_layer15_30.jsonl

训练代码：train_tiny_transformer.py

训练集大小：500000

输入空间大小估计： $2^{30} = 1073741824$

训练集大小占输入空间大小的比例：0.0466%

任务相关设计思想和讨论：

1.为了严格检验神经网络超越模式识别、学习并执行精确符号规则的能力，我选择了一维元胞自动机作为首要的实验平台。这一选择并非偶然，而是基于其独特的、能够剔除一切模糊性的理想特性。首先，CA是一个完全由本地化、确定性规则驱动的计算系统，其行为不依赖于任何源自现实世界的统计先验或语义信息，这使得它成为测试纯粹符号操纵能力的完美“无菌环境”。其次，其巨大的状态空间（对于一个宽度为W的二进制元胞自动机，存在 2^W 个可能的状态）使得任何形式的“死记硬背”都变得徒劳，从而强制模型去学习规则本身，而非记忆输入输出对。

在此基础上，我特别选择了Rule 110作为核心测试规则。因为Rule 110不仅是一个展现复杂和看似混沌行为的非平凡规则，更关键的是，它已被证明是图灵完备的。这意味着，原则上，Rule 110能够模拟任何计算过程。因此，让Transformer学习Rule 110的演化，就等同于在测试模型是否具备学习一种通用计算的潜力。成功地学习这一规则，将为“神经网络能够内化复杂算法”这一论点，提供最坚实、最令人信服的实验证据。

2.一些任务参数：

宽度：30

规则：110

层数：15

实验结果与分析：评估损失收敛至0.000068。这个极低的评估损失和训练集大小远小于输入空间的事实充分表明神经网络已经学会这个规则。

最终收敛结果：

--- Step 477000 ---

Train Loss: 0.001766

Eval Loss: 0.000068

4.1.2. 学习本质探究：N进制加法与语义洗牌

任务描述：模型学习预测和输出两个N进制数的加法，同时对比语义洗牌和位置洗牌的训练结果

输入格式：16位的字符串，前8位代表一个数，后8位代表一个数

输出格式：33个标签的多标签二分类格式，等同于33个01字符串，用33位二进制数表示加法结果

loss：平均二元交叉熵损失

训练配置：4090显卡

训练集生成脚本：generate_symbol_add_shuffle_dataset.py

数据集：adder_8bit_base16_train.jsonl——无语义洗牌无位置洗牌

adder_8bit_base16_sem_shuffled_train.jsonl——有语义洗牌无位置洗牌

adder_8bit_base16_pos_shuffled_train.jsonl——无语义洗牌有位置洗牌

adder_8bit_base16_sem_shuffled_pos_shuffled_train.jsonl——有语义洗牌有位置洗牌

训练代码：train_tiny_transformer.py

训练集大小：500000

输入空间大小估计： $16 \times 16 = 256$ 次方 = $18446744073709551616 = 1.84e19$

训练集大小占输入空间大小的比例： $2.71e-14$

任务相关设计思想和讨论：

1. 模型学习对两个8位16进制数进行加法运算。为了探究模型学习的本质，我设计了“语义洗牌”对照实验，包括两种洗牌，一种是用随机不同符号表示16进制数，一种是16进制数的位置排列随机选择，不再采用前8位后8位分别是两个16进制数的做法。

这样设计实验的原因：

- 语义洗牌：确定transformer学习的是相对于任务的符号之间的结构，和特定符号是什么无关。
- 位置洗牌：确定transformer学习这个任务，以及绝大多数任务，不依赖于输入的符号排列方式。

2. 一些任务相关参数和细节：

进制数：16进制

两个数的各自位数：8

无语义shuffle的情形下：0123456789abcde作为16进制数符号表示

有语义shuffle的情形下：任取不同的可见ascii字符作为16进制数符号表示

无位置shuffle的情形下：输入字符串的前8位是一个8位16进制数，后8位是另一个

有位置shuffle的情形下：在训练数据集脚本的开始，产生一个位置shuffle的映射，然后在整个数据集保持这个映射的一致，相应代码如下

```
if SHUFFLE_POSITIONS:
```

```
    position_map = list(range(INPUT_LEN))
```

```
    random.shuffle(position_map)
```

```
    POSITION_SHUFFLE_MAP = {from_idx: to_idx for from_idx, to_idx in
                           enumerate(position_map)}
```

实验结果与分析：下表的结果决定性地证明了我的核心主张。在“无语义洗牌”和“有语义洗牌”两种情况下，模型最终都收敛到了极低的损失水平 (<0.0001)，且性能无显著差异。这无可辩驳地表明，模型学习到的是加法运算背后抽象的代数结构，而非表面的符号模式。

注：不同随机数据和训练的随机种子，产生的训练loss下降过程可能相差很多，所以不能通过单一一次对比分析训练难易程度。但是在多次尝试中，发现4种实验设置的收敛性是稳定存在的。

由于训练过程较短，在此列出所有4个实验的训练过程，以eval loss表示。

step	无语义shuffle无位置shuffle	无语义shuffle有位置shuffle	有语义shuffle无位置shuffle	有语义shuffle有位置shuffle
1000	0.534105	0.505196	0.511109	0.564834
2000	0.438036	0.263462	0.446237	0.497787
3000	0.347195	0.001700	0.365586	0.461389
4000	0.324581	0.001392	0.179969	0.395561
5000	0.254971	0.000123	0.063128	0.328988
6000	0.129849	0.000042	0.001083	0.247512
7000	0.066946	0.000024	0.000203	0.159110
8000	0.044260		0.000123	0.149018
9000	0.032533		0.000051	0.139221
10000	0.001005			0.103275
11000	0.000325			0.080524
12000	0.000099			0.052849
13000				0.022913
14000				0.002443
15000				0.002063
16000				0.000327
17000				0.000472
18000				0.000272
19000				0.000293
20000				0.000409
21000				0.000173
22000				0.000164
23000				0.000391
24000				0.000243
25000				0.000095

4.2. 高级能力：算法学习与规划

4.2.1. 案例：接雨水问题

任务描述：源自LeetCode经典算法题，[42. 接雨水 - 力扣（LeetCode）](#) [20]

42. 接雨水

困难

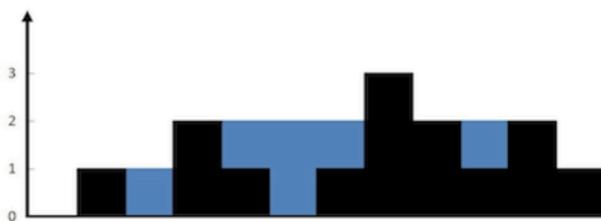
相关标签

相关企业

Ax

给定 n 个非负整数表示每个宽度为 1 的柱子的高度图，计算按此排列的柱子，下雨之后能接多少雨水。

示例 1：



输入: height = [0,1,0,2,1,0,1,3,2,1,2,1]

输出: 6

解释：上面是由数组 [0,1,0,2,1,0,1,3,2,1,2,1] 表示的高度图，在这种情况下，可以接 6 个单位的雨水（蓝色部分表示雨水）。

示例 2：

输入: height = [4,2,0,3,2,5]

输出: 9

提示：

- `n == height.length`
- `1 <= n <= 2 * 104`
- `0 <= height[i] <= 105`

通过次数 1,421,469 / 2.2M | 通过率 65.6%

输入格式：30位的01字符串，每连续3位01字符串，以3位二进制数格式代表一个柱子的高度（0-7），总共十根柱子

输出格式：30位的01字符串，每连续3位01字符串，以3位二进制数格式代表一个柱子的所接雨水量（0-7），总共十根柱子

loss：平均二元交叉熵损失

训练配置：4090显卡

训练集生成脚本：generate_trapping_rain_water_decoupled.py

数据集：trapping_rain_water_decoupled_n10_b3_train.jsonl

训练代码：train_tiny_transformer.py

训练集大小：300000

输入空间大小估计：2的30次方=1073741824=1.07e9

训练集大小占输入空间大小的比例：0.0279%

任务相关设计思想和讨论：

1.leetcode是一个测试这个范式的算法能力的天然仓库。我从这里测试了很多算法题，很多题这个范式都是可以拟合的。关于接雨水这道题，它在leetcode上属于困难题。其解法也不是任何简单的符号规则可以模拟的。至于拟合的原理，应该说是暂时未知的。但是我认为transformer并没有按人类通常的思维认为的那样，先解析格式，再执行一步步的算法步骤。因为它不可能通过任何方法知道每一位符号的意义，而且这个任务的含义它也不可能知道，它只是在执行一个简单的任务，按loss定义的目标随机梯度下降。这也是我认为这个范式可以用——用联结主义的方式，梯度下降去逼近离散——描述的原因。

2.关于为什么输出不是接雨水的总量，因为我之前直接把总量的二进制数作为目标训练，发现效果并不好，收敛困难。于是我采用了这种解耦的格式，得以极其迅速地收敛。在这里，我给transformer减少了一个求和的负担，这应该是模型迅速收敛的原因。另外，我做的另一个实验，也曾因为串行混合了较多的简单运算导致收敛困难，这是个很有趣的现象，会在后面讨论。

3.一些任务参数：

柱子个数：10

表示柱子高度和接雨水两所用的二进制数位数：3

柱子高度范围/接雨水量范围：0-7

实验结果与分析：经过16,000步训练，模型评估损失收敛至0.000059。这一结果表明，本范式能够有效学习需要理解全局信息（如左右最高墙）的复杂算法，并通过输出格式的解耦，实现了对问题结构的精确建模。

最终收敛结果：

--- Step 16000 ---

Train Loss: 0.000570

Eval Loss: 0.000059

4.2.2. 稠密迷宫寻路

任务描述：在一个复杂的、为人类玩家设计的稠密迷宫中，模型学习预测从任意位置出发，通往终点的最优路径的“下一步”方向（上/下/左/右）。

输入格式：169位的由“0”或者“1”或者“s”或者“t”组成的字符串，表示迷宫的初始状态，每连续的13个字符表示迷宫的一行，0表示空地，1表示墙，s表示起始点，t表示目标点，s和t都有且仅有一个，在13*13迷宫的外围默认有一圈墙。

输出格式：4分类

loss：四分类交叉熵损失

训练配置：4090显卡

训练集生成脚本：generate_maze_dense.py

数据集：maze_optimized_13_13_dataset.jsonl

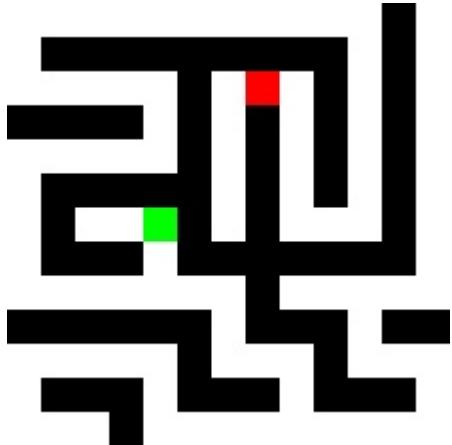
训练代码：train_tiny_transformer.py

训练集大小：2000000

输入空间大小估计：采用蒙特卡洛法估计2000000次无重复样本，暂且粗糙地估计为2的169次方=7.48e50

训练集大小占输入空间大小的比例：约2.673e-45

以下图表示一个典型的迷宫，用白色代表空地，用黑色代表墙壁，用绿色代表起点，用红色代表终点。这是一个典型的适合人类玩的迷宫游戏，以区别于随机产生独立墙壁点的迷宫生成算法。



任务相关设计思想和讨论：

1. 基于对于这个范式的算法拟合能力的信心，这个迷宫实验的设计一开始的算法其实相当于随机产生墙壁点。但是实践证明，这样的迷宫，对于人类来说可能非常容易，可以一眼看出最短路径（再次也是可行路径），但对于模型来说，分支数和复杂度都远超目前的迷宫实验设置。
2.之所以选择最短路径的下一步方向，是因为这种格式设计起来比较简单，直接输出最短路径的话，无法提前知道最短路径的长度，即使采用其他措施，也使得这个任务设计变得不太优雅。另外一个重要原因是，这样的设计意味着另一种可能，迷宫任务可以被看成是在某个 state 选择最佳 action 的任务，这个可以和强化学习联系起来，这样训练 state-action 对，可以试这个范式有完成强化学习任务的潜力，见后面对 arc-agl-3 的讨论。训练也有直接监督学习最佳 action 和强化学习探索最佳 action 两种方法，但限于时间精力我并没有完成直接用强化学习探索最佳 action 的实验。

3.一些任务参数：

宽度：13

高度：13

实验结果与分析：评估损失收敛至 0.000014，模型已展现出极强的路径规划能力。这证明了本范式可以将传统的“搜索”问题，摊销进一个前馈网络的权重中，实现高效的“无搜索规划”。这个极低的评估损失意味着模型已经学会走类似的迷宫，而极低的评估损失也意味着在这个最佳路径的长度尺度上，整个路径准确率也会非常高。

最终收敛结果：

--- Step 279000 ---

Train Loss: 0.000083

Eval Loss: 0.000014

4.3. 从图像中推理和提取符号输出

本节将展示该范式在处理多模态任务时的强大潜力，包括从视觉信息中解码符号

4.3.1. 视觉符号解码：表盘角度识别

任务描述：图像推理，通过输入图像类似表盘的东西，输出这些不同颜色和粗细的线段对应的角度占用所有角度区间的情况

输入格式：224*224图像

输出格式：36个标签的多标签二分类格式，等同于36个01字符串，表示各个区间是否有被线段占用，36个区间平分360度角，每个区间10度角

loss：平均二元交叉熵损失

训练配置：4090显卡

训练集生成脚本：generate_line_angle_to_vector.py

数据集：line_angle文件夹

训练代码：train_swin_image2text.py

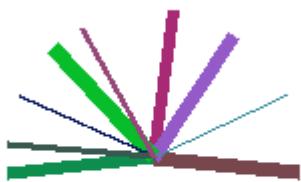
使用模型：Swin Transformer

训练集大小：200000

输入空间大小估计：389329651

训练集大小占输入空间大小的比例：0.0514%

图示：



任务相关设计思想和讨论：

1. 算法拟合能力得到证明后，我联想到transformer在图像中的应用，于是想测试图像transformer有没有类似的推理能力。于是用ViT测试了棋盘格相关的任务，效果很好。再之后就设计了这个任务，初衷是想测试纯粹的图像理解，而非简单识别图像中的符号再应用transformer已经被证明的符号规则学习能力和算法拟合能力。

2. 这个任务一开始采用了ViT模型，效果不好，于是后面换成了现在所使用的Swin Transformer。

3. 为了增加多样性，线段的颜色和粗细加入了随机性

4. 一些任务参数：

图像分辨率：224*224

区间个数：36

每个区间角度：10

实验结果与分析：评估损失收敛至0.019183。这个loss并不算完美收敛，但也证明模型能够从原始像素中，精确地“解码”出抽象的、符号化的角度信息，展现了强大的视觉-符号接地能力。

最终收敛结果：

--- Step 24000 ---

Train Loss: 0.016392

Eval Loss: 0.012753

4.4. 图像推理能力

本节将展示该范式纯粹的图像推理能力，以和符号规则学习能力和算法拟合能力区分。

4.4.1. 几何推理与构造：预测三角形内切圆

任务描述：模型的输入是一张随机生成的绿色三角形图像，任务是生成其在数学上唯一的、正确的红色内切圆图像。

输入格式：224*224图像

输出格式：224*224图像

loss: MSELoss

训练配置：4090显卡

训练集生成脚本：generate_triangle_to_incircle.py

数据集：incircle_dataset文件夹

训练代码：train_image2image.py

使用模型：Swin Transformer+unet

训练集大小：150000

输入空间大小估计：采用蒙特卡洛法估计约 $2.8e+12$

训练集大小占输入空间大小的比例：约 $5.4e-8$

任务相关设计思想和讨论：

1.为了进一步检验transformer纯粹的图像推理能力，把它和符号规则学习能力和算法拟合能力分开。我设计了这个内切圆实验。当然理论上来说识别3个点的位置再采用一系列数学公式计算出内切圆的位置，再画出内切圆，理论上存在可能，但实践中几乎是不可能是真实的过程。因此这个任务的作用是为了证明transformer的图像推理能力。鉴于图像的模拟性质，这种推理能力很有向更大的领域扩展的潜能，比如纯粹的实验观测数据之类，但是很容易想到的是，鉴于真实的实验观测数据一定是有噪声的，又会退化到之前的模式识别深度学习范式，这个问题会在后面讨论。

2.这个任务和后面相关的系列任务最惊人的地方是以非常直观的形式展现了“机器心智”在训练中的演化，模型是采用什么方式去逼近和学习规则的，在此过程中表现出的特征非常类似于某种智能。训练过程的可视化以eval image形式展现。

3.一些任务参数：

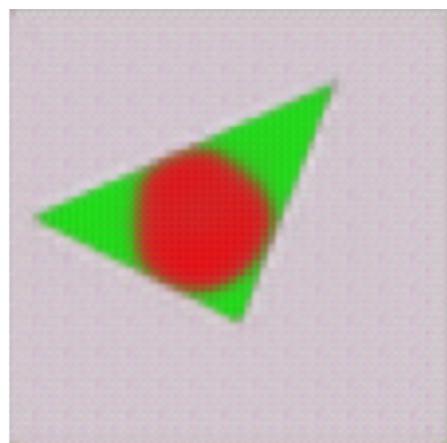
图像分辨率：224*224

实验结果与分析：最终的训练结果表明模型已经完全学会内切圆的规则，这表明模型具有图像推理能力，在这个例子确实不能排除内插其他训练集例子的可能，但是后面的其他任务又排除了这种可能，因为那些任务无法用内插解释，而它们使用完全相同的模型和超参数配置。同时，模型的学习速度非常快，提供了直接观察机器心智的形成过程。

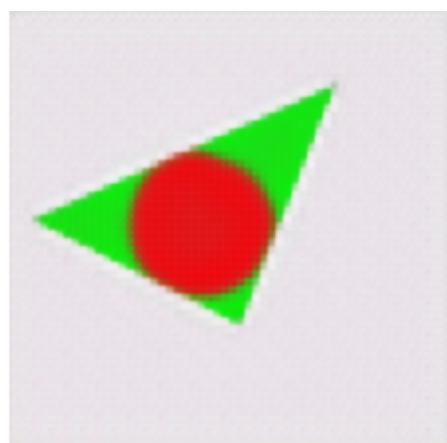
实验过程展示：

左边是输入图像，右边是ground truth，中间是生成图像。

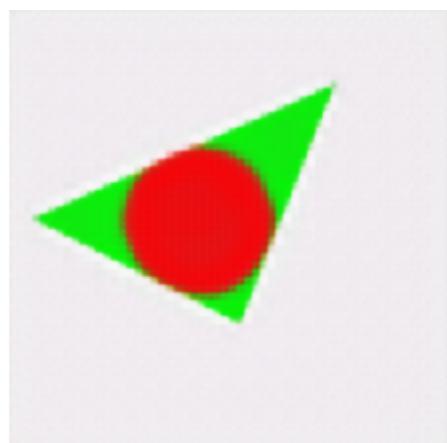
200 step



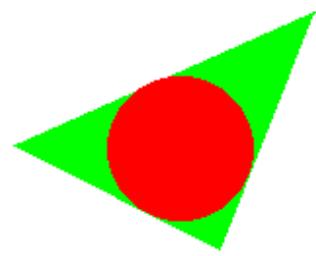
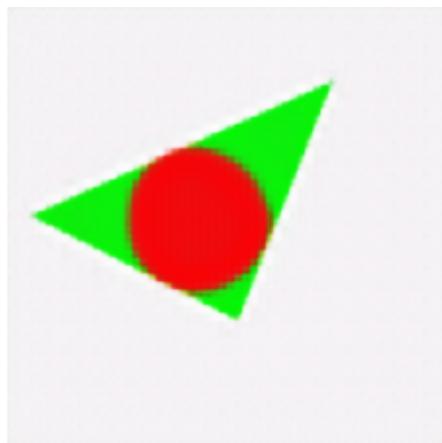
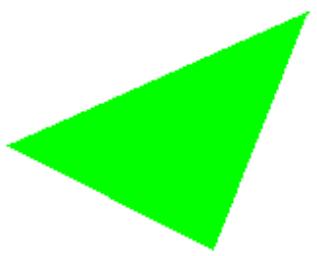
400step



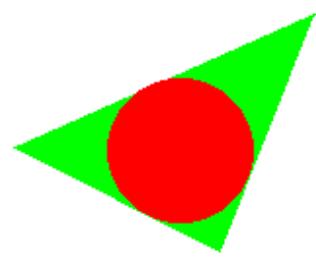
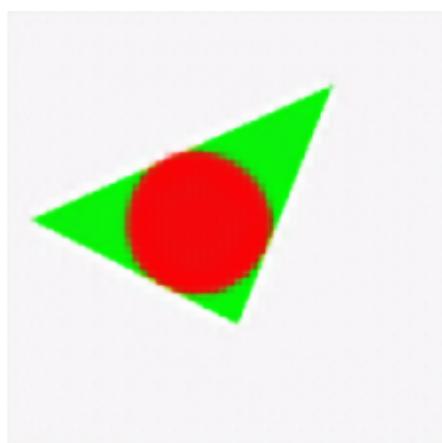
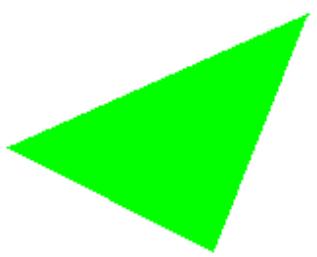
600step



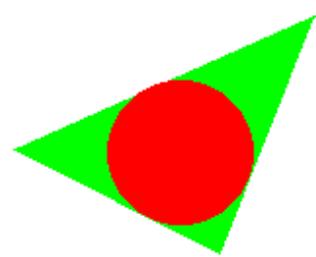
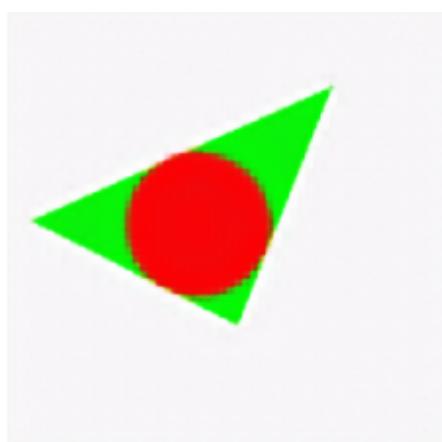
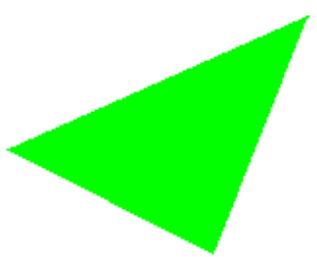
800step



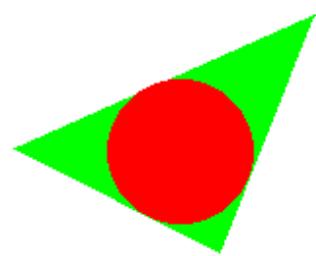
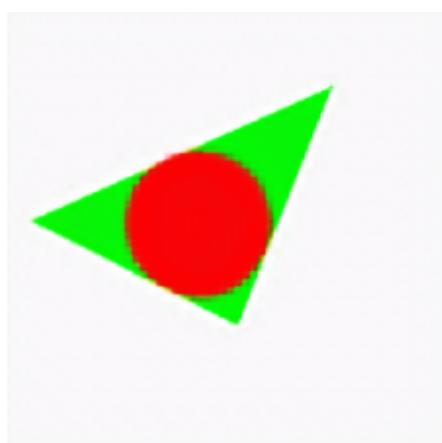
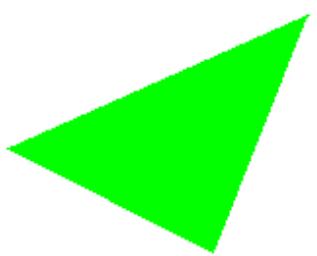
1000step



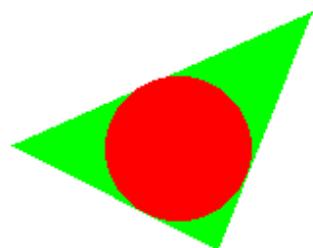
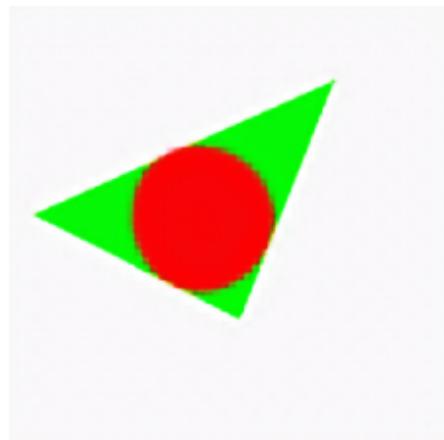
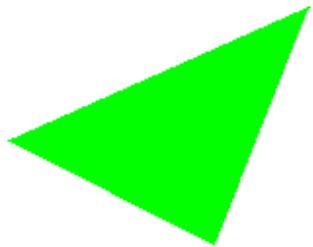
1200step



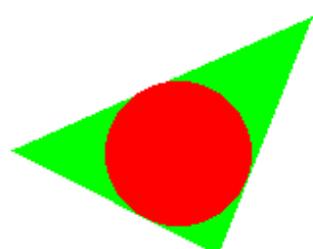
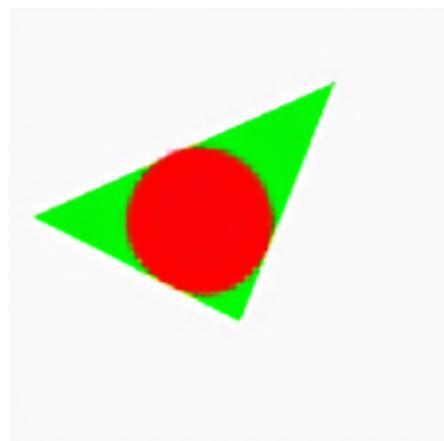
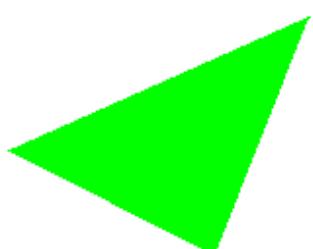
1400step



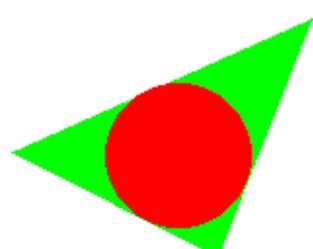
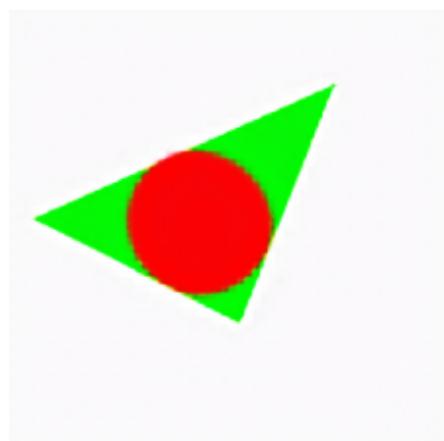
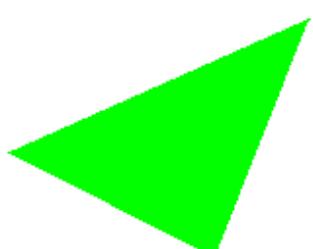
1600step



1800step

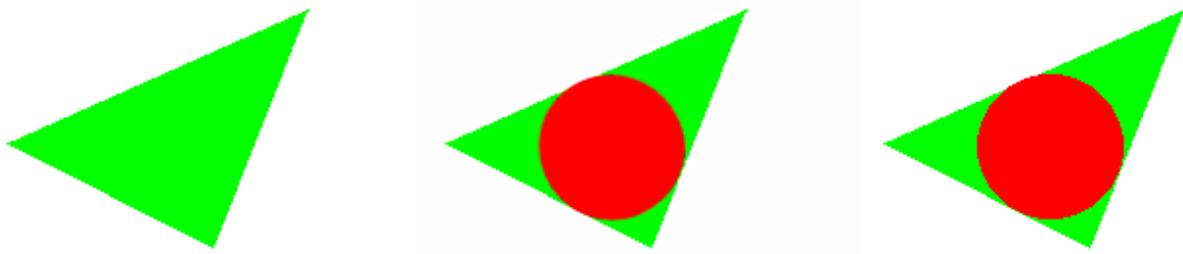


2000step



后面由于变化速度减缓，直接展示最后训练结果

14000step



4.4.2. 几何推理与构造：平面镶嵌

任务描述：模型的输入是一张随机生成的绿色三角形图像，输出图像是绿色三角形的平面镶嵌，绿色三角形和红色三角形边相邻接和交替

输入格式：224*224图像，包含一张随机生成的绿色三角形图像

输出格式：224*224图像，为输入图像的平面镶嵌结果，绿色三角形和红色三角形边相邻接和交替

loss: MSELoss

训练配置：4090显卡

训练集生成脚本：generate_triangle_to_tessellation.py

数据集：tessellation_dataset_224文件夹

训练代码：train_image2image.py

使用模型：Swin Transformer+unet

训练集大小：150000

输入空间大小估计：采用蒙特卡洛法估计约 $2.71e10$

训练集大小占输入空间大小的比例：约 $5.5e-6$

任务相关设计思想和讨论：

1.这个任务可以完全排除以内插来解决任务而不是学习精确规则的可能，原因是，在平面镶嵌的远处，这种内插的误差会逐渐扩大，以至于无法再通过朴素的内插方法拟合规则

2.为了确保任务难度适中，对输入绿色三角形的形状和大小进行了限制，详见训练集生成脚本代码

3.一些任务参数：

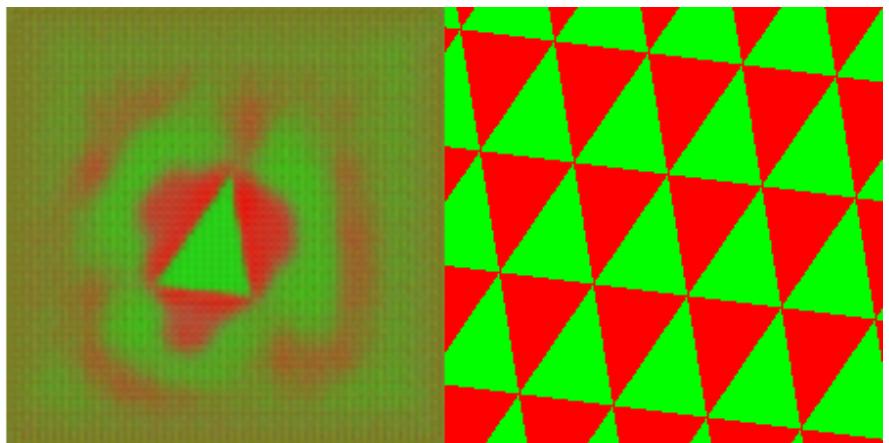
图像分辨率：224*224

实验结果与分析：最终的训练结果表明模型已经完全学会平面镶嵌的规则，这表明模型具有图像推理能力，且很大程度上排除了模型在进行朴素内插的可能。同时，模型提供了直接观察机器心智的形成过程，可以看出这个例子中显现出一个非常明显地由输入三角形周围向外部扩展的学习过程。

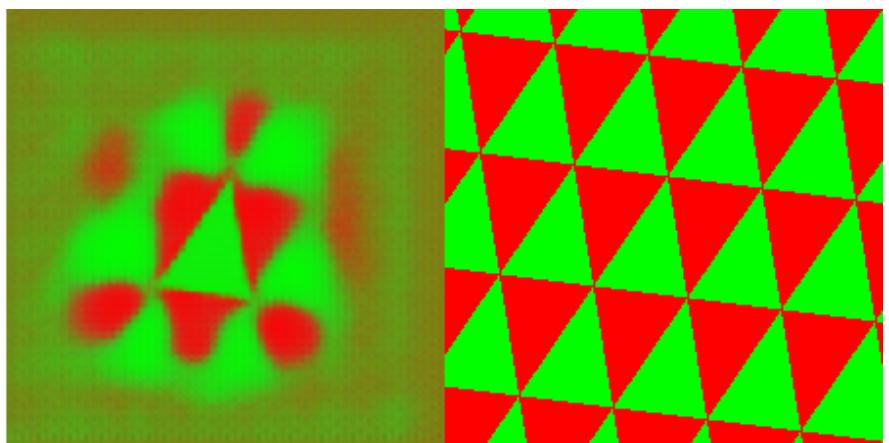
实验过程展示：

左边是输入图像，右边是ground truth，中间是生成图像。

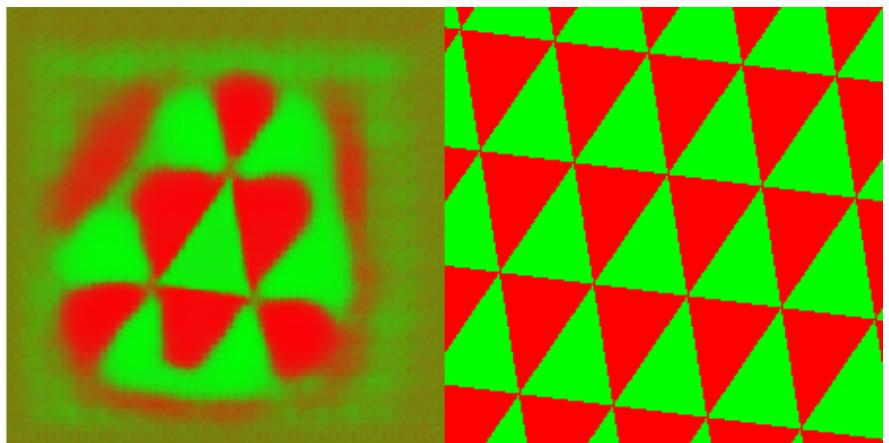
200step



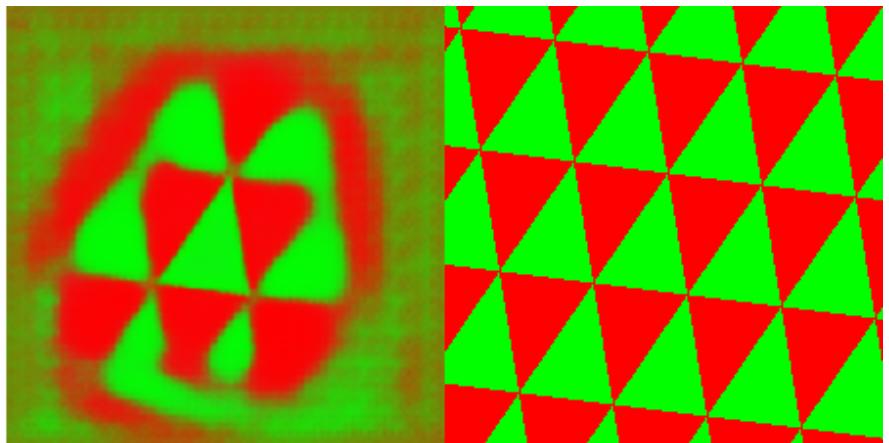
400step



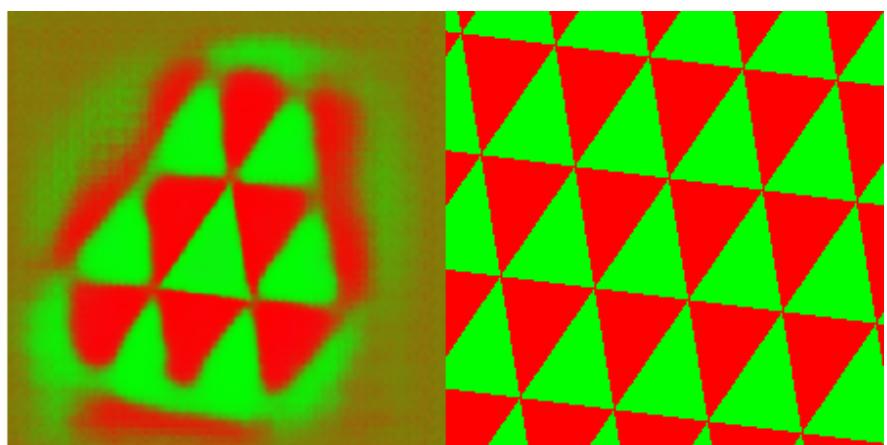
600step



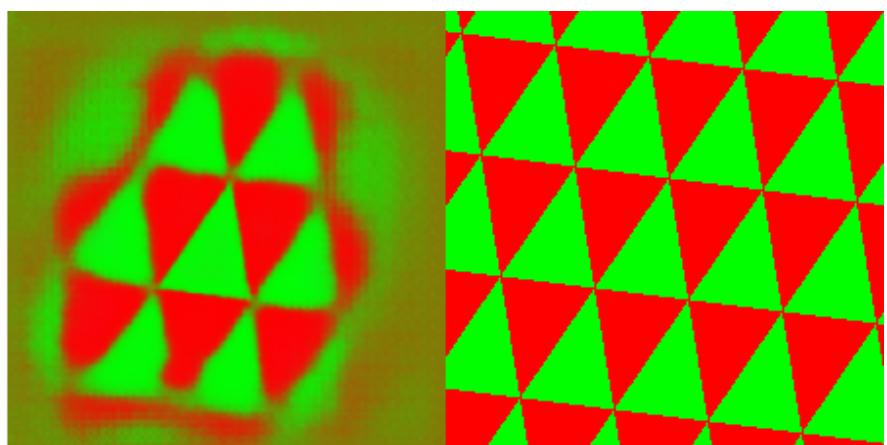
800step



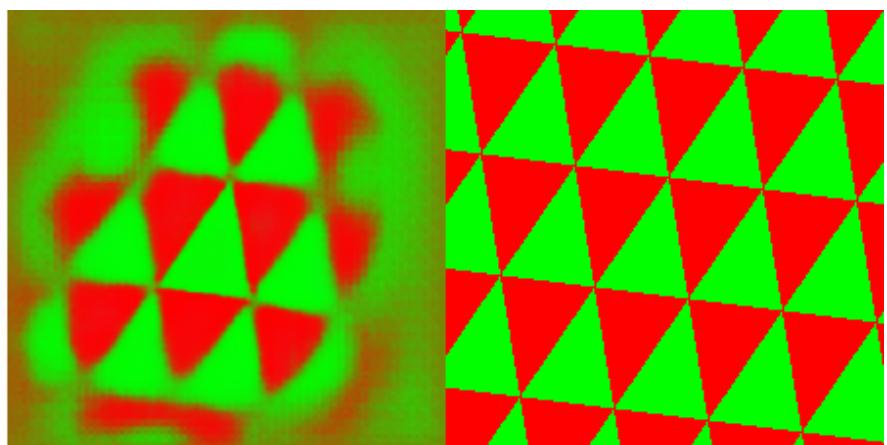
1000step



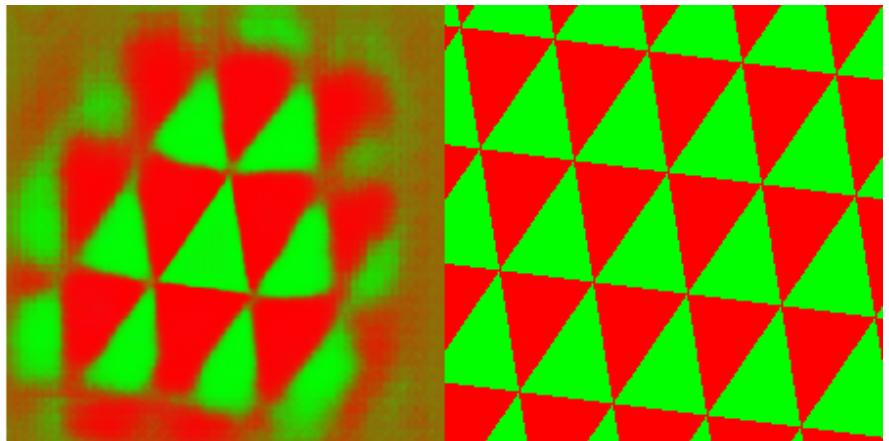
1200step



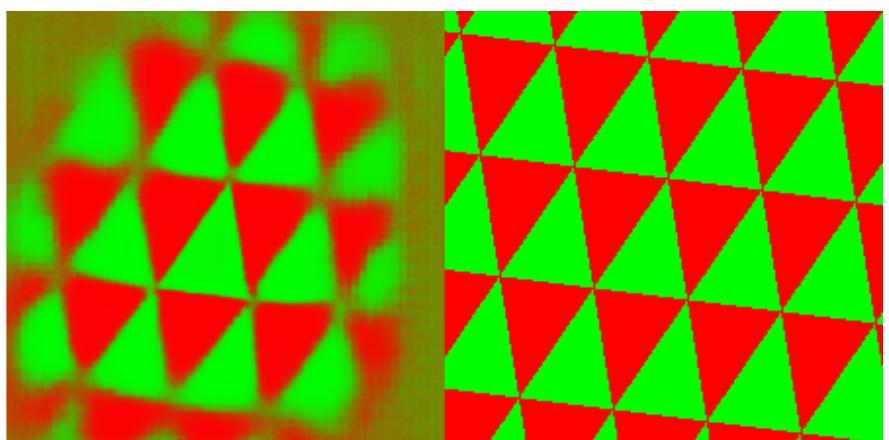
1400step



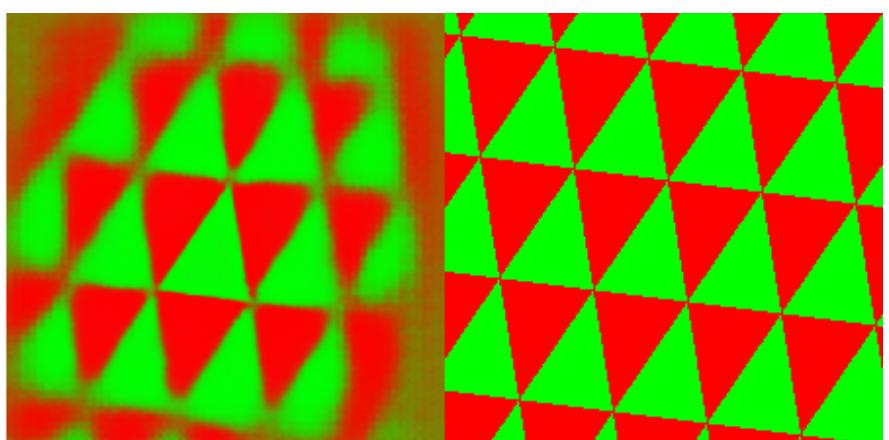
1600step



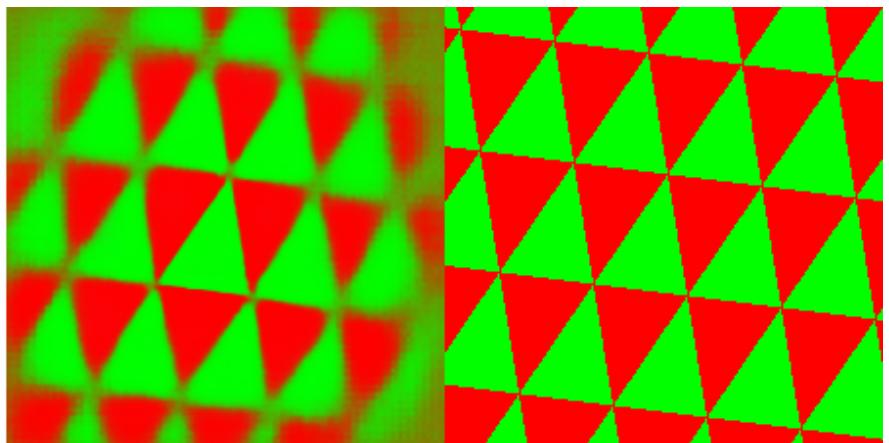
1800step



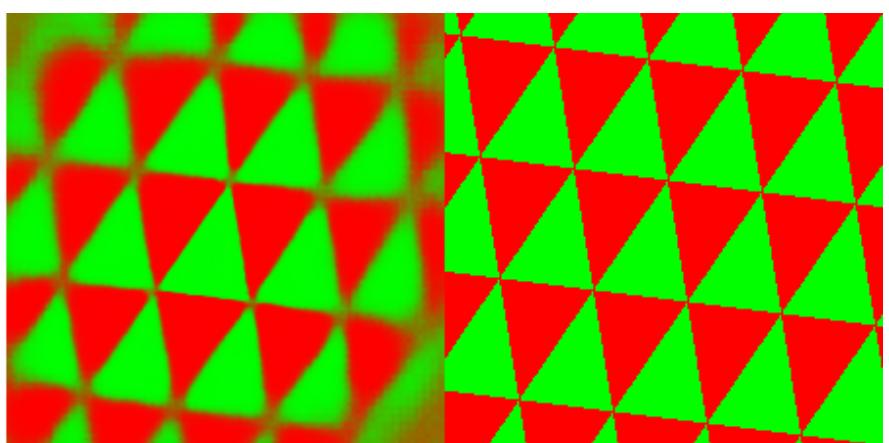
2000step



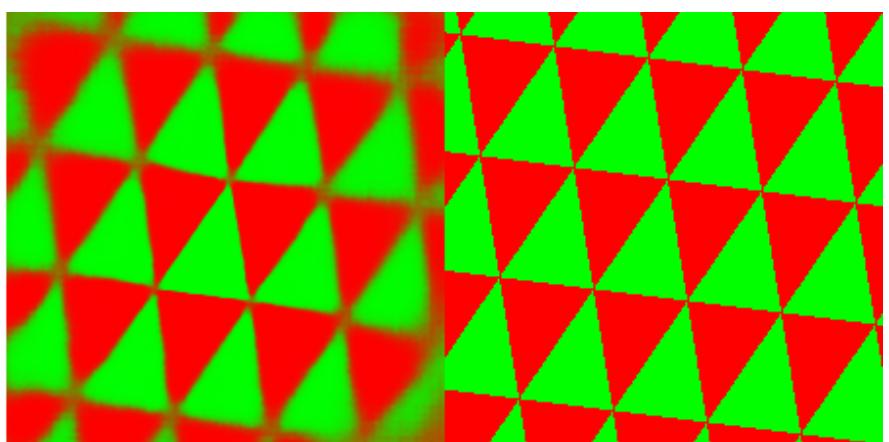
3000step



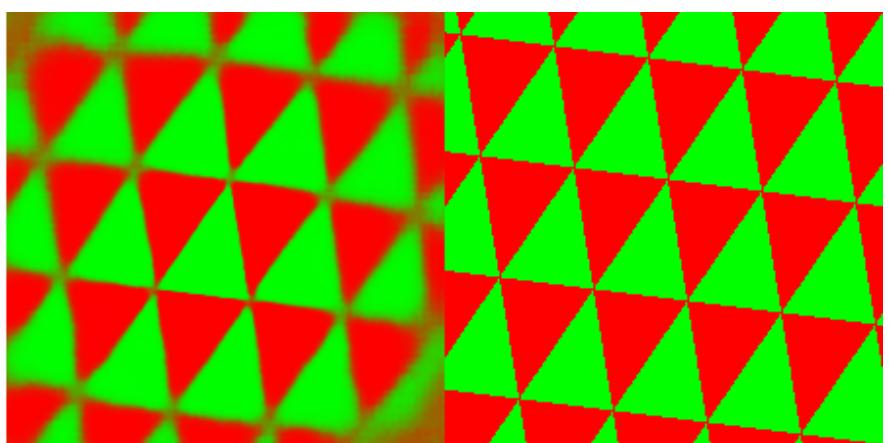
4000step



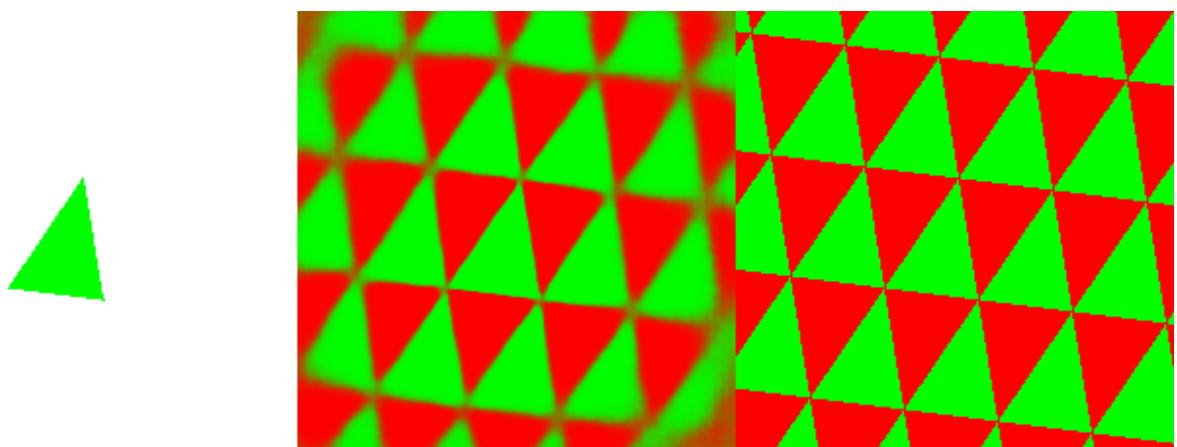
5000step



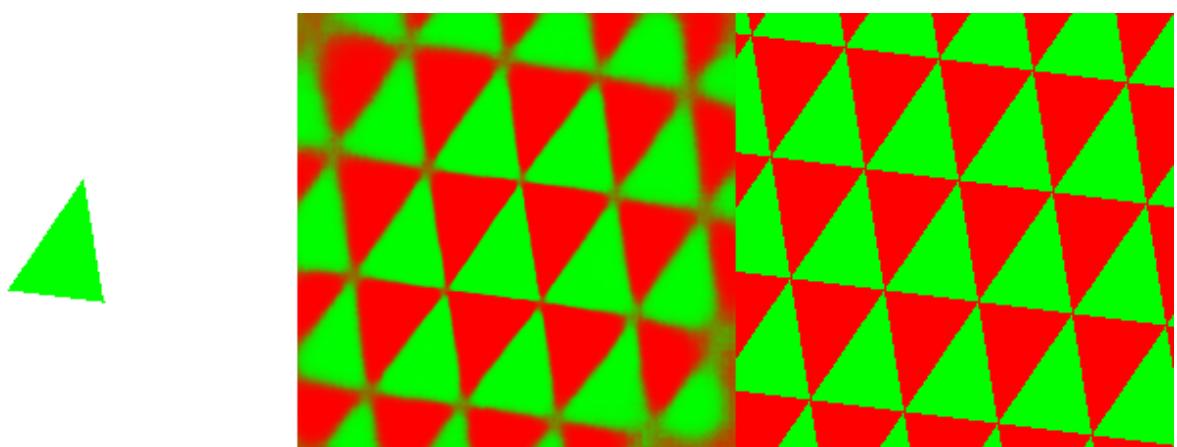
6000step



7000step

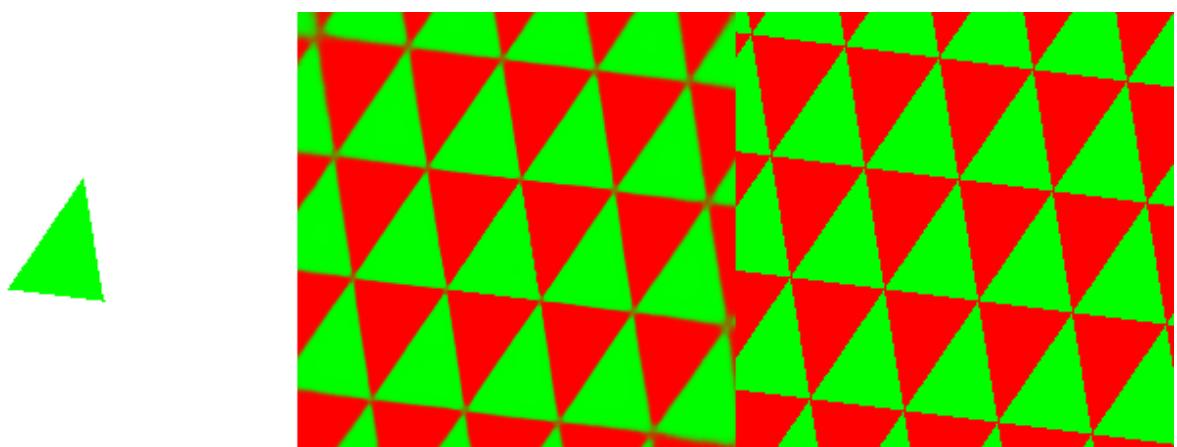


8000step



后面由于变化速度减缓，直接展示最后训练结果

150000step



4.5. 符号推理生成图像

本节将展示该范式在处理符号输入，学习对应规则，生成对应图像的能力。

4.5.1. 元胞自动机

任务描述：模型学习预测和输出一个一维元胞自动机给定规则变换给定次数后的状态

输入格式：36位的由“0”或者“1”组成的字符串，表示元胞自动机的初始状态

输出格式：240*240图像

loss: MSELoss
训练配置: 4090显卡
训练集生成脚本: generate_cellular_automata_1d_to_grid_image.py
数据集: ca_render_dataset_240文件夹
训练代码: train_qwen2_text2image.py
使用模型: transformer+unet
训练集大小: 300000
输入空间大小估计: 2的36次方=68719476736
训练集大小占输入空间大小的比例: 4.4e-6

任务相关设计思想和讨论:

1.这个任务是为了检验transformer符号生成图像的推理能力，这个模式并非传统意义上的文生图，而是重点检验它的推理能力。

2.输出图像中，把一维36位的元胞自动机状态以行优先排列成6*6的棋盘格图像，黑色代表1，白色代表0

3.一些任务参数:

格点维度: 6*6

格点大小: 40*40像素

图像分辨率: 240*240

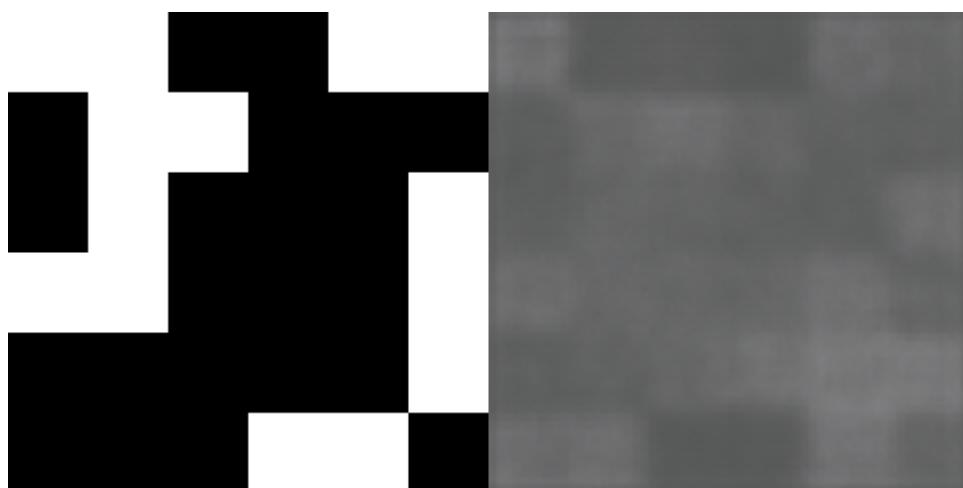
元胞自动机规则: rule 110

规则演化层数: 3

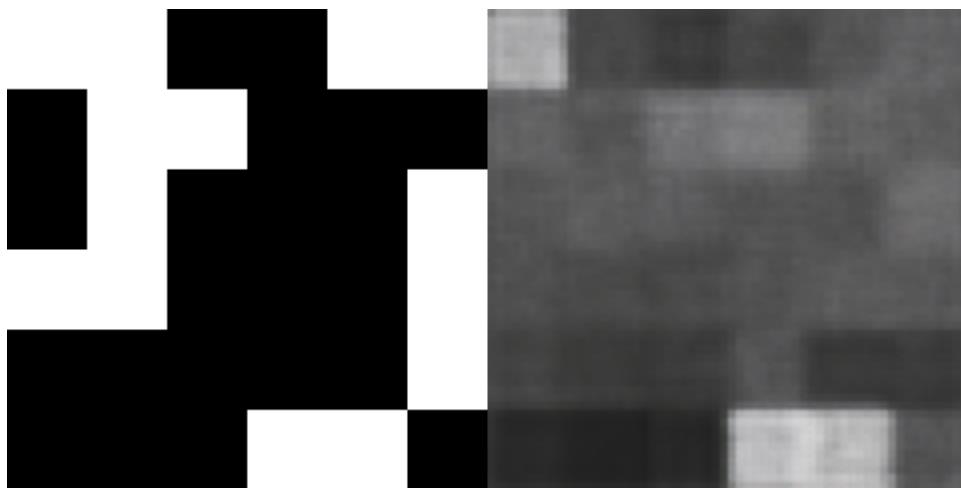
实验结果与分析：最终的训练结果表明模型已经完全学会元胞自动机的规律，这表明，这种符号生图的模式，也可以保留transformer的推理能力。模型的学习速度非常快，提供了直接观察机器心智的形成过程，整个学习过程无法简单用某种逻辑解释，更进一步证明这种范式的学习不一定遵循人们所以为的逻辑，而是更加朴素的以联结主义的逼近离散的方式。

左边是ground truth，右边是模型生成图像。

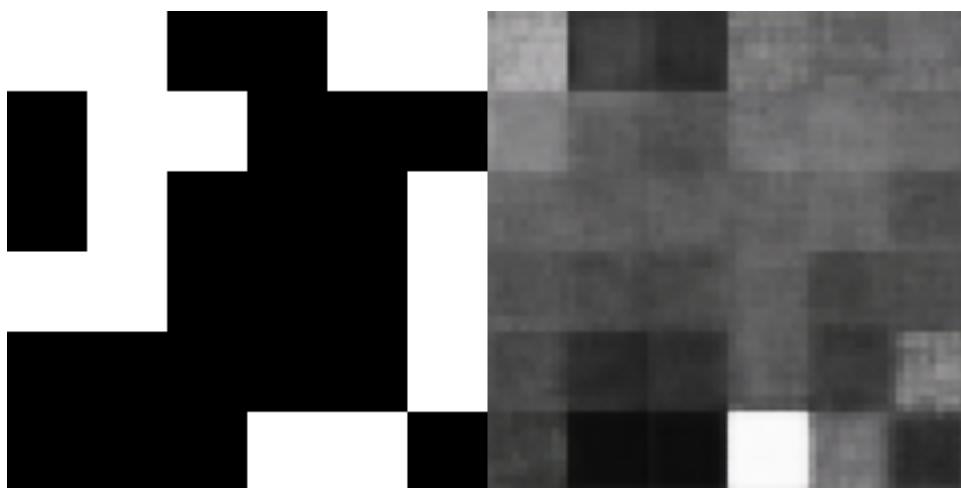
100 step 符号输入: 111110011010001000111010000110



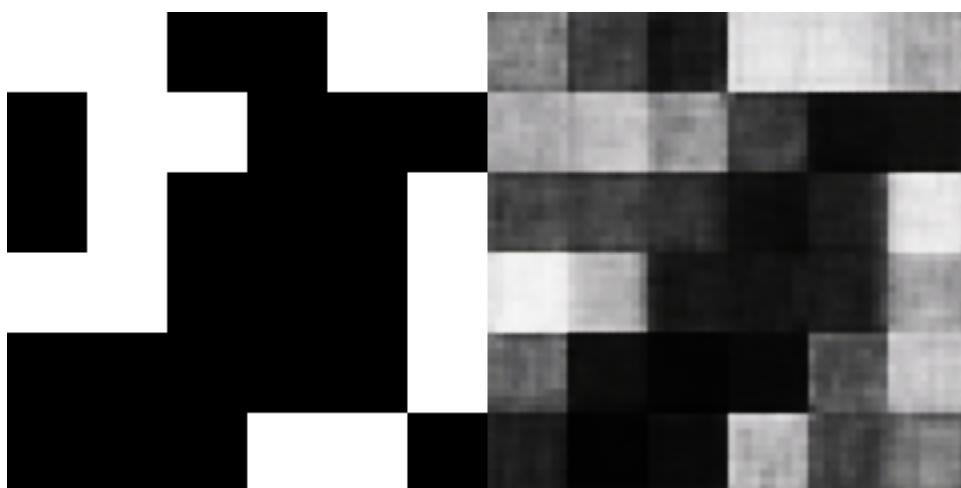
200 step 符号输入: 111110011010001000111010000110



300 step 符号输入：111110011010001000111010000110



400 step 符号输入：111110011010001000111010000110



500 step 符号输入：111110011010001000111010000110



600 step 符号输入：111110011010001000111010000110



700 step 符号输入：111110011010001000111010000110



800 step 符号输入：111110011010001000111010000110



4.5.2 旋转立方体

任务描述：模型学习预测和输出根据旋转角，生成一个彩色六方体旋转后的图像

输入格式：24位的由“0”或者“1”组成的字符串，表示3个旋转角，连续8位二进制数表示一个旋转角

输出格式：256*256图像

loss: MSELoss

训练配置：4090显卡

训练集生成脚本：generate_cube_rotation_pillow_with_anchor.py

数据集：cube_dataset_final_highlight文件夹

训练代码：train_qwen2_text2image.py

使用模型：transformer+unet

训练集大小：300000

输入空间大小估计：16777216

训练集大小占输入空间大小的比例：1.79%

任务相关设计思想和讨论：

1.在这个任务中遇到了一个特殊的困难，即模型很难把旋转角对应的顶点和符号输入联系起来，导致无法收敛。解决方法是，永远在生成训练集图片的最后步骤高亮一个特殊的顶点，不考虑遮挡关系。采取这样的解决方法后，模型最终得以在这个任务上最终收敛。在另外一个任务上，我用符号指示三角形的位置，输出平面镶嵌结果，模型遇到了同样的困难，它无法了解该把符号对应到输出图像的哪一个绿色三角形。解决方法类似，我把符号输入指示的三角形在输出图像中的颜色改为了黑色，一切就正常了。这应该是比较典型的问题，即模型很难确定如何把符号输入对应输出图像的哪一个特征，解决方案应该也是类似的。

2.关于每个面上的白色十字线段，只是为了美观，可能也有一些辅助训练的作用。

3.符号输入的3个旋转角，即pitch, yaw, roll，具体对应请参考训练集生成代码。

4.其实我有另一个版本的模型，采用的不是lora微调qwen2-0.5b的方法，而是以一个比qwen2-0.5b的总参数更少，但比qwen2-0.5b的lora参数更多的一个transformer，实践证明效果不如qwen2-0.5b的学习效果，本文认为这是一个有意思的现象。

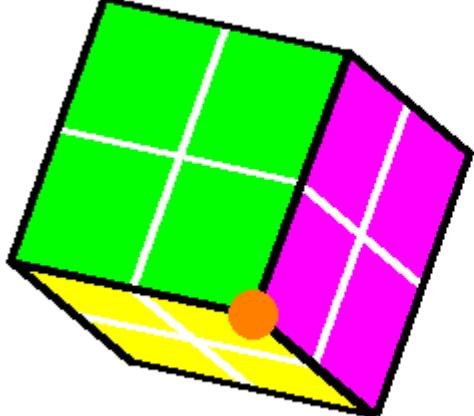
5.一些任务参数：

图像分辨率：256*256

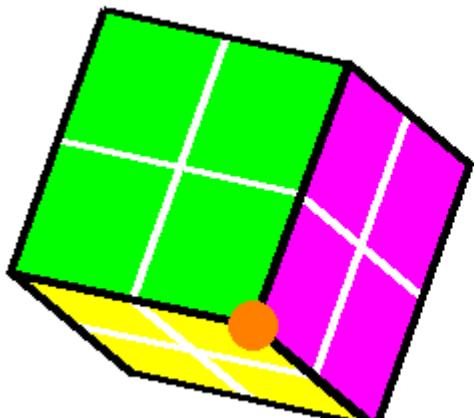
实验结果与分析：最终的训练结果表明模型已经完全学会这个任务，这表明，这种符号生图的模式，也可以保留transformer的图像推理能力，而鉴于我使用的是qwen2-0.5b的lora微调，可以说这种图像推理能力，至少是生成图像的能力，不限于Swin Transformer，ViT等图像transformer。模型的学习速度并不是很快，可能也是因为任务相对较难，提供了直接观察机器心智的形成过程，在这个任务上可以看到，一开始模型还没有理解旋转角，只是简单学会了生成圆形和较为平均的填充颜色，在后面模型慢慢理解了旋转立方体的概念，再后面模型慢慢填充精确的细节。

左边是ground truth，右边是模型生成图像。

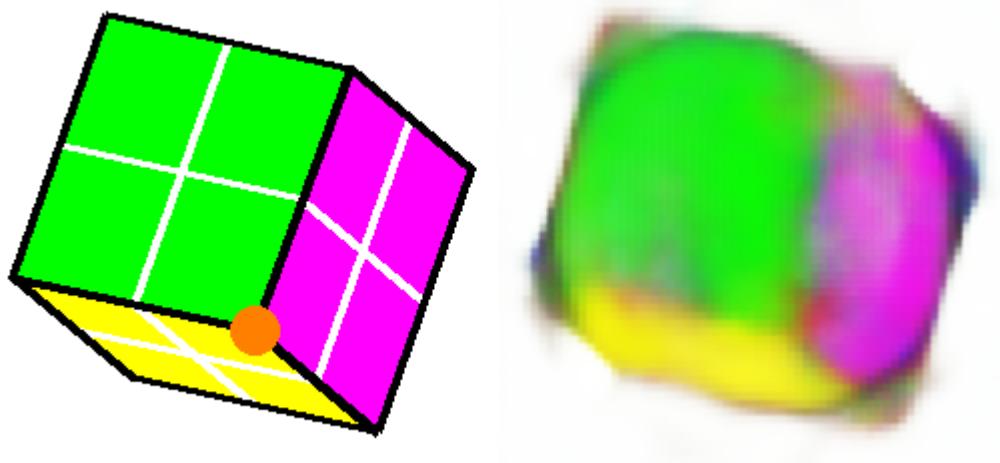
step 500 符号输入：100101101000100100110010



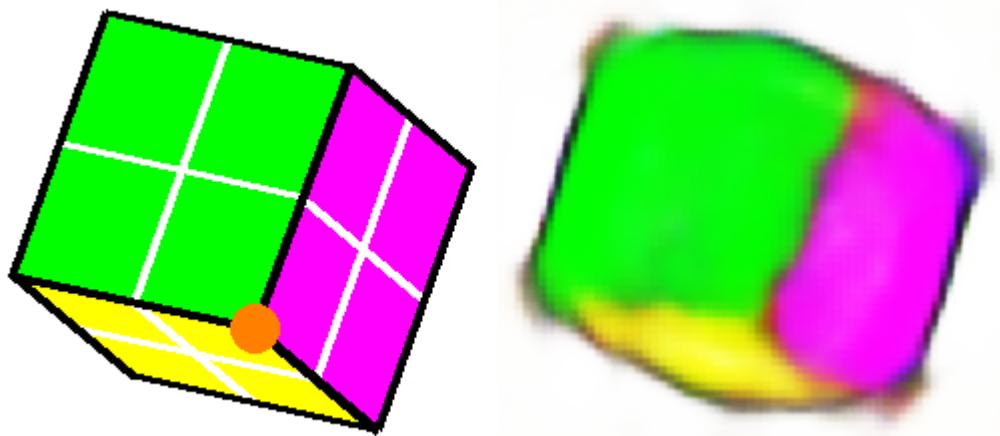
step 1000 符号输入：100101101000100100110010



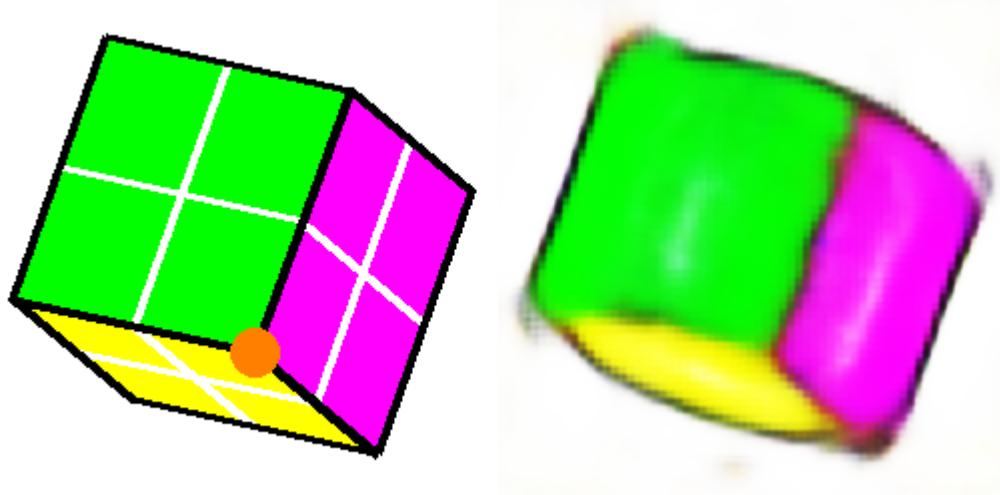
step 1500 符号输入：100101101000100100110010



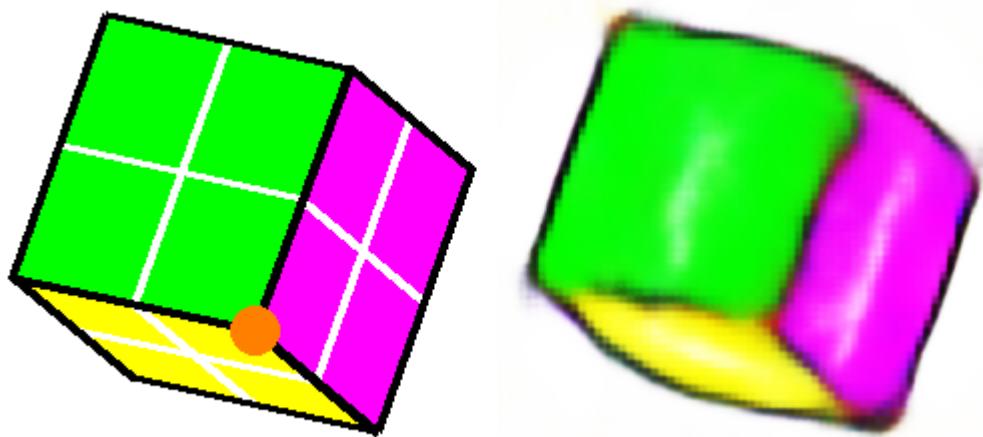
step 2000 符号输入：100101101000100100110010



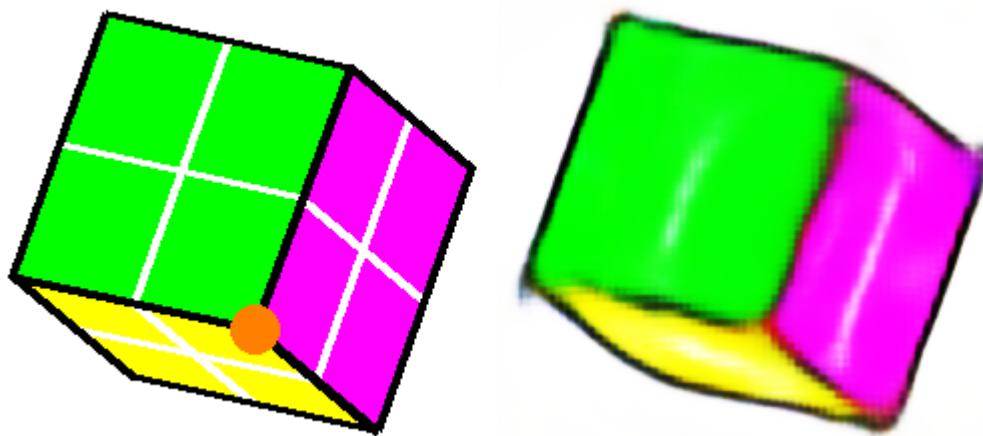
step 2500 符号输入：100101101000100100110010



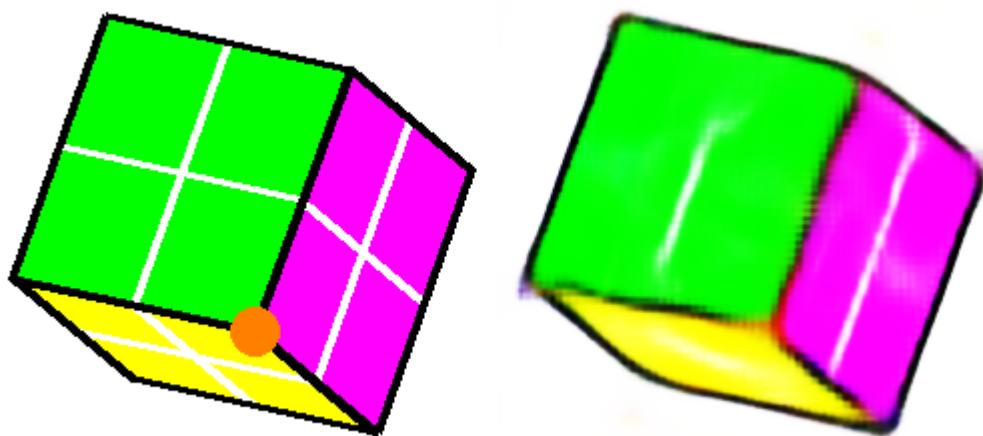
step 3000 符号输入：100101101000100100110010



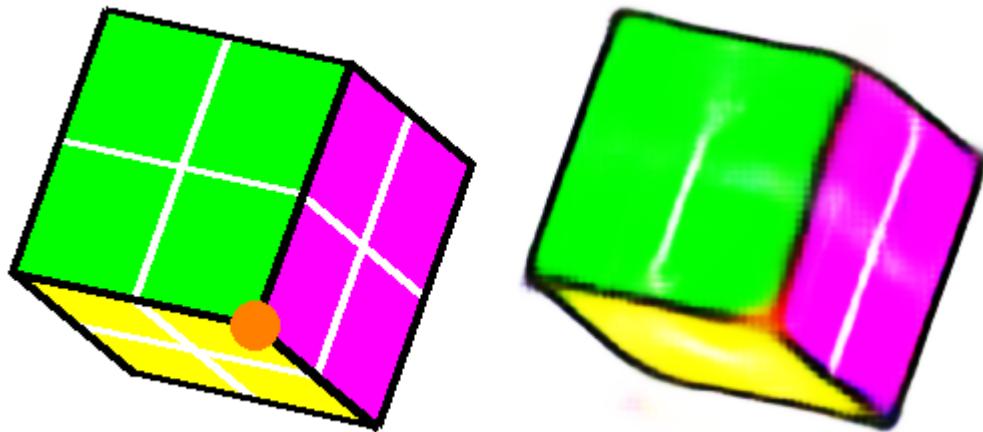
step 4000 符号输入：100101101000100100110010



step 6000 符号输入：100101101000100100110010

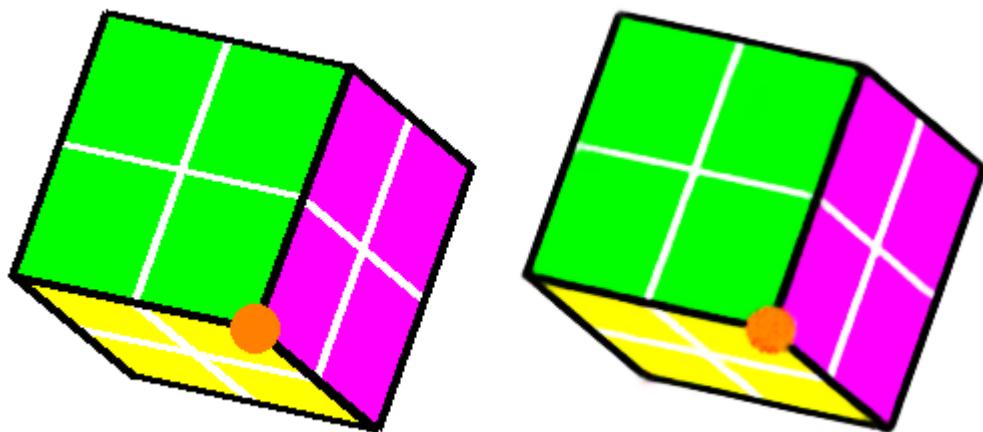


step 8000 符号输入：100101101000100100110010



后面由于变化速度减缓，直接展示最后训练结果

step 133600 符号输入：100101101000100100110010



4.6. 物理模拟

本节将展示该范式在模拟物理过程中的能力。

4.6.1 悬链线

任务描述：给定悬链线两端的端点，和悬链线上的某一个给定的经过点，模型学习绘制出由物理定律（最小势能原理）所唯一确定的悬链线轨迹。

输入格式：224*224图像

输出格式：224*224图像

loss：MSELoss

训练配置：4090显卡

训练集生成脚本：generate_catenary_curve_from_points.py

数据集：catenary_dataset_v5_CONSTRUCTIVE文件夹

训练代码：train_image2image.py

使用模型：Swin Transformer+unet

训练集大小：150000

输入空间大小估计：采用蒙特卡洛法估计约 $9.3e9$

训练集大小占输入空间大小的比例：约 $1.61e-5$

任务相关设计思想和讨论：

- 1.这个任务是为了检验这个范式的物理相关的模拟能力，通过图像推理模型实现。
- 2.虽然这个问题有通过内插解决的可能性，但是使用的模型和超参数配置同时解决了不可能由内插解决的其他问题，从而排除了这种可能性。
- 3.之所以加入悬链线经过点这个限制，主要是因为设置悬链线为固定长度或者通过某种方式表示悬链线长度的方式感觉都不如这种设置更好。然后加入悬链线经过点这个限制，也增加了输入空间，进一步排除记忆或内插的可能性。

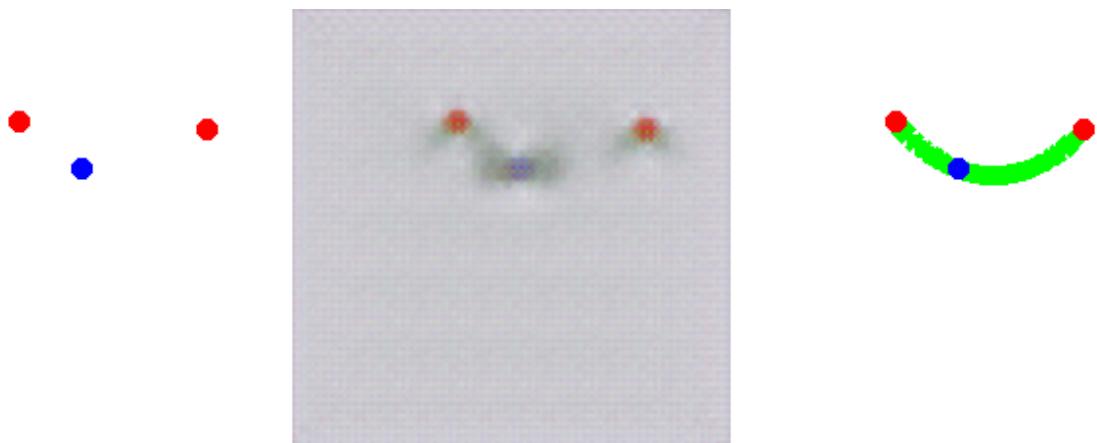
4.一些任务参数：

图像分辨率：224*224

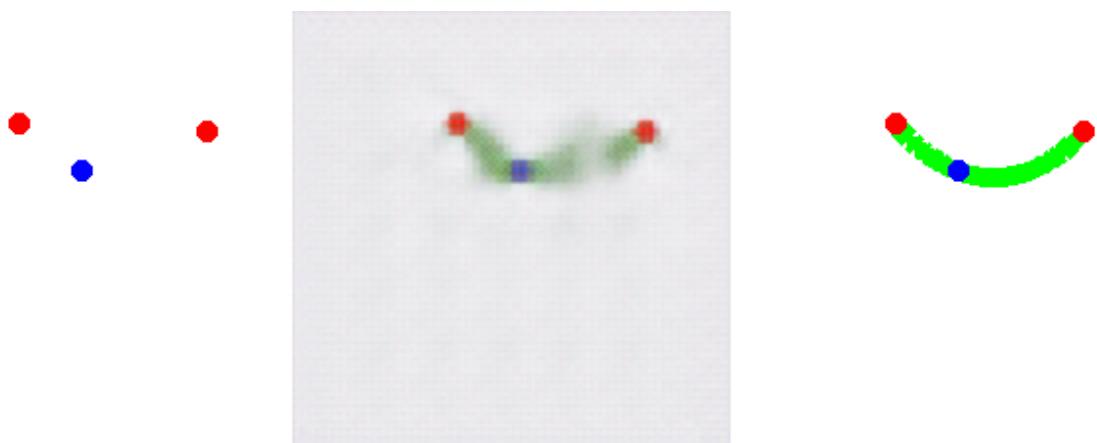
实验结果与分析：最终的训练结果表明模型已经完全学会悬链线的规律，这表明，本范式的学习能力，已从纯粹的数学和逻辑法则，扩展到了对我真实宇宙基本物理规律的模拟，但同时这种模拟的精度需要进一步探究。模型的学习速度非常快，提供了直接观察机器心智的形成过程，可以看出这个例子中显现出的学习过程并没有在悬链线的轨迹上花太长时间，而是主要在优化图像输出。同时注意到我产生的训练集是有瑕疵的，输出悬链线图像有毛边，而模型收敛后的输出却是相对平滑的，这个现象值得进一步研究。

左边是输入图像，中间是模型生成图像，右边是ground truth。

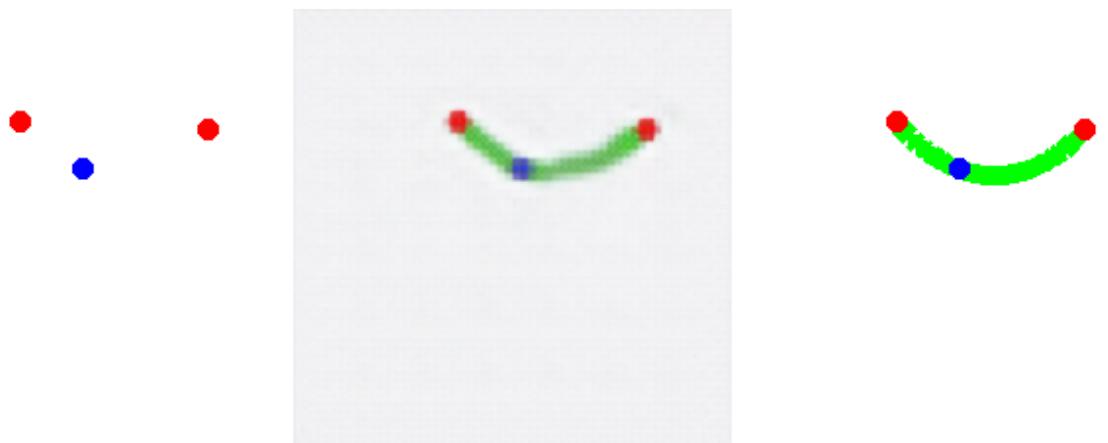
200step



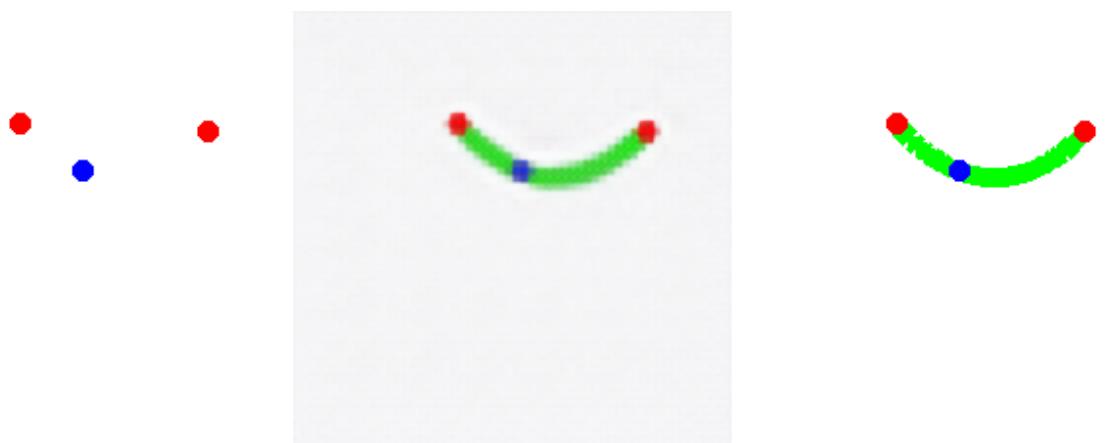
400step



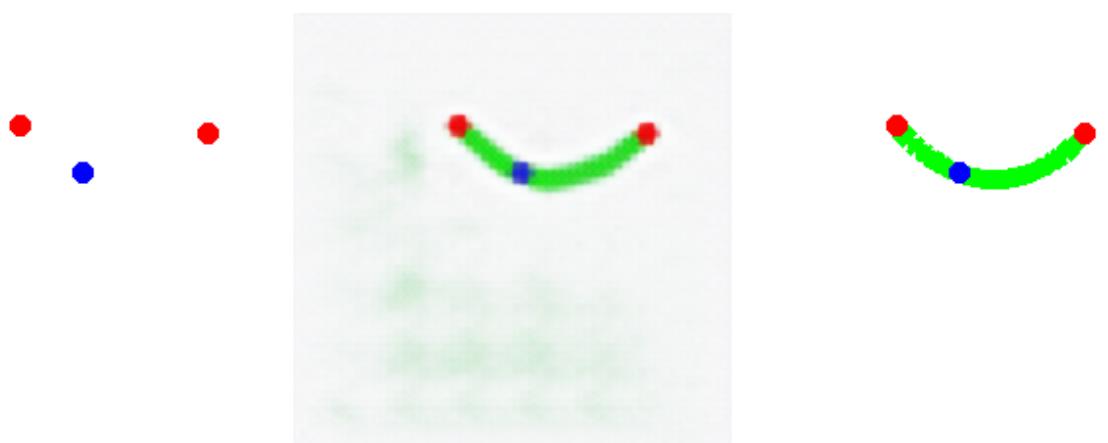
600step



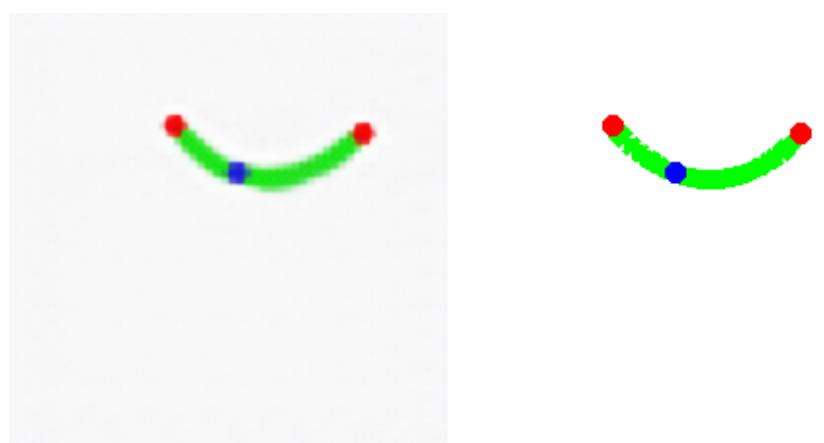
800step



1000step



1200step



1400step



1600step



1800step

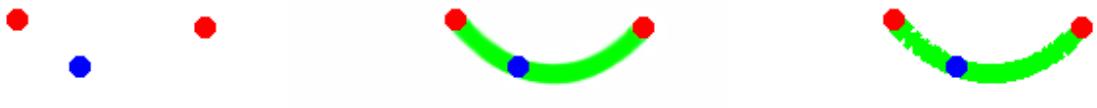


2000step



后面由于变化速度减缓，直接展示最后训练结果

19000step



4.6.2. 行星轨道模拟

任务描述：给定恒星的固定位置，行星初始位置，初始速度大小，初始速度方向，预测行星轨道

输入格式：224*224图像

输出格式：224*224图像

loss: MSELoss

训练配置：4090显卡

训练集生成脚本：generate_orbital_path_from_initial_state.py

数据集：orbital_dataset_256_separated_v3文件夹

训练代码：train_image2image.py

使用模型：Swin Transformer+unet

训练集大小：150000

输入空间大小估计：采用蒙特卡洛法估计约 $2.8e11$

训练集大小占输入空间大小的比例：约 $5.36e-7$

任务相关设计思想和讨论：

- 1.这个任务是为了进一步检验这个范式的物理相关的模拟能力，通过图像推理模型实现。
- 2.虽然这个问题有通过内插解决的可能性，但是使用的模型和超参数配置同时解决了不可能由内插解决的其他问题，从而排除了这种可能性。
- 3.恒星的固定位置，行星的初始位置相对容易在输入图像中表示，关于行星的初始速度大小和

方向，我采用这种方式表示：从行星初始位置发出一个较短的射线线段，用颜色代表速度大小，线段方向代表速度方向，关于速度大小和颜色对应方式见训练集生成脚本代码。

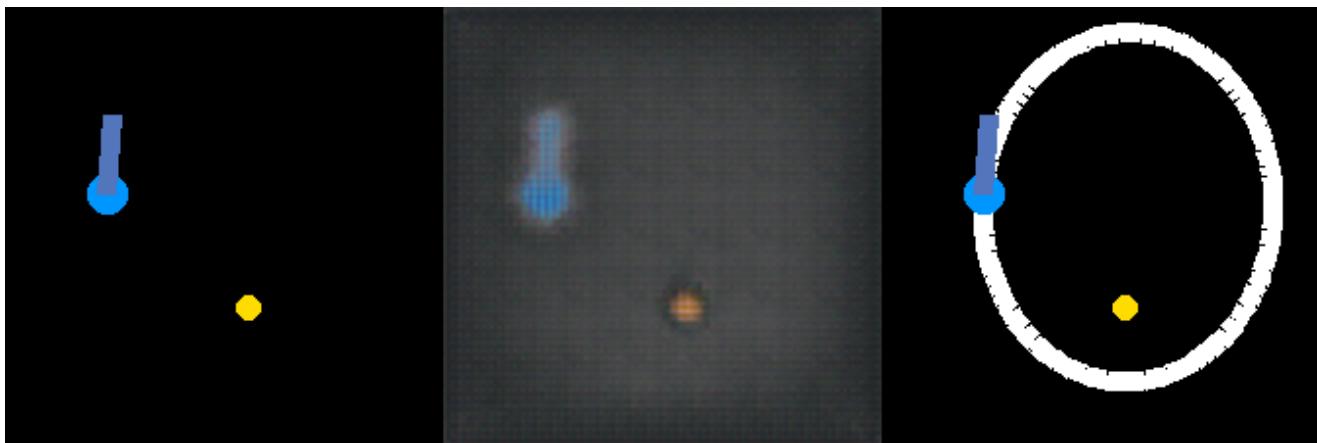
4.一些任务参数：

图像分辨率：224*224

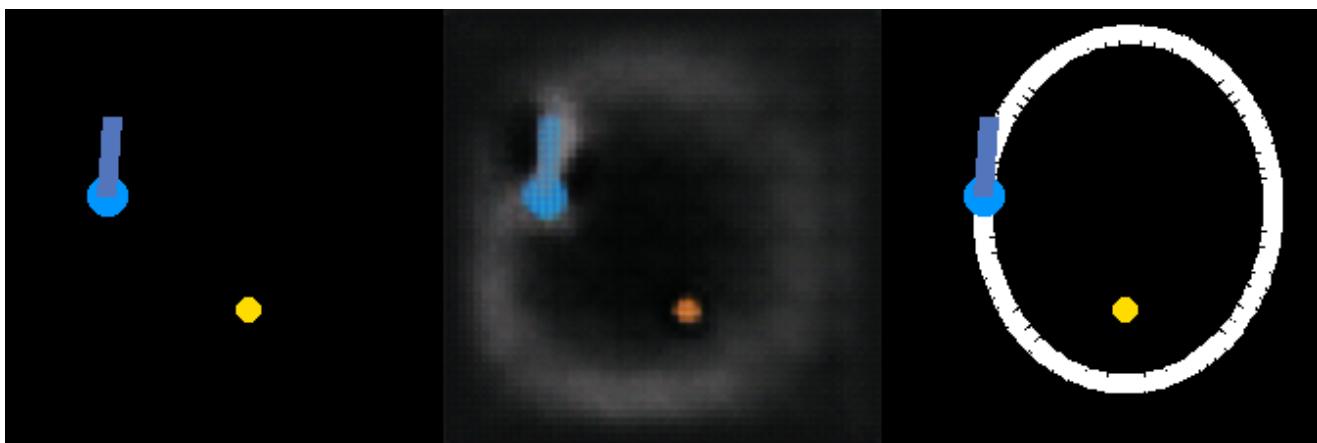
实验结果与分析：最终的训练结果表明模型已经完全学会行星轨道相关的规律，但同时这种模拟的精度需要进一步探究。模型提供了直接观察机器心智的形成过程，这个任务的学习过程显现了一个有趣的现象，由于输出图像的轨道设置为白色，在学习的过程中，尤其在学习的初期，这种白色弥散在输出图像中，令人联想到量子力学中的概率云。

左边是输入图像，中间是模型生成图像，右边是ground truth。

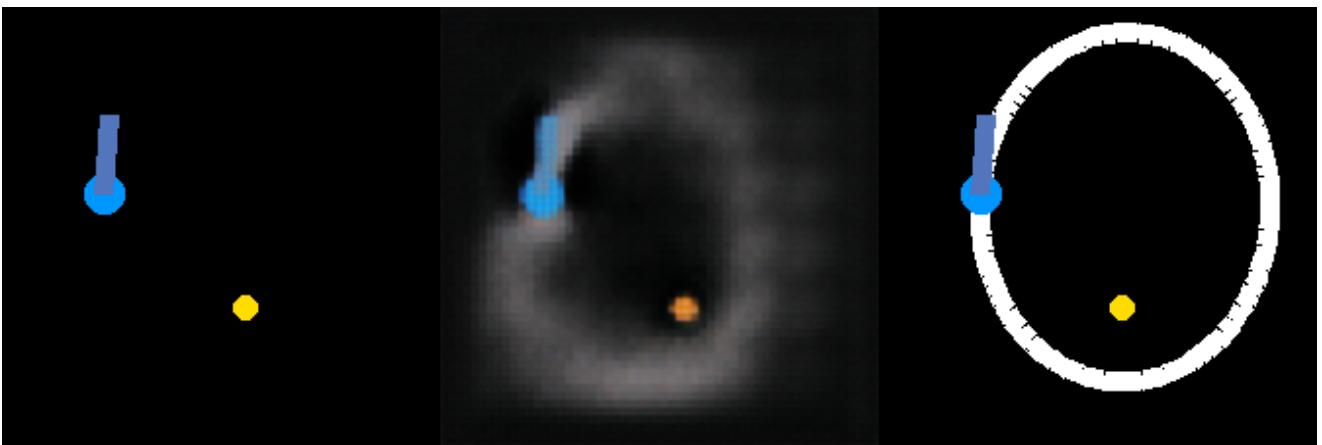
200step



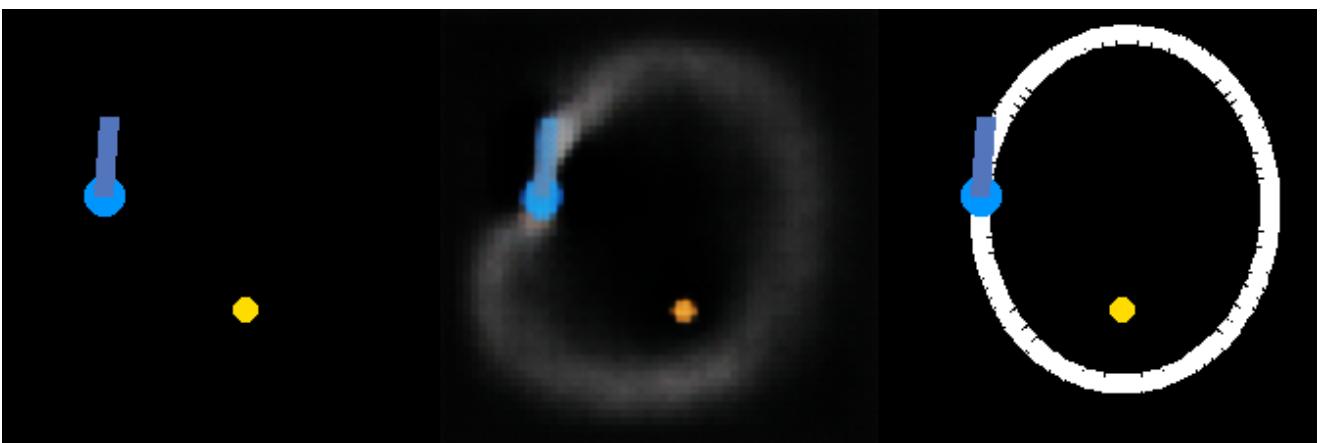
400step



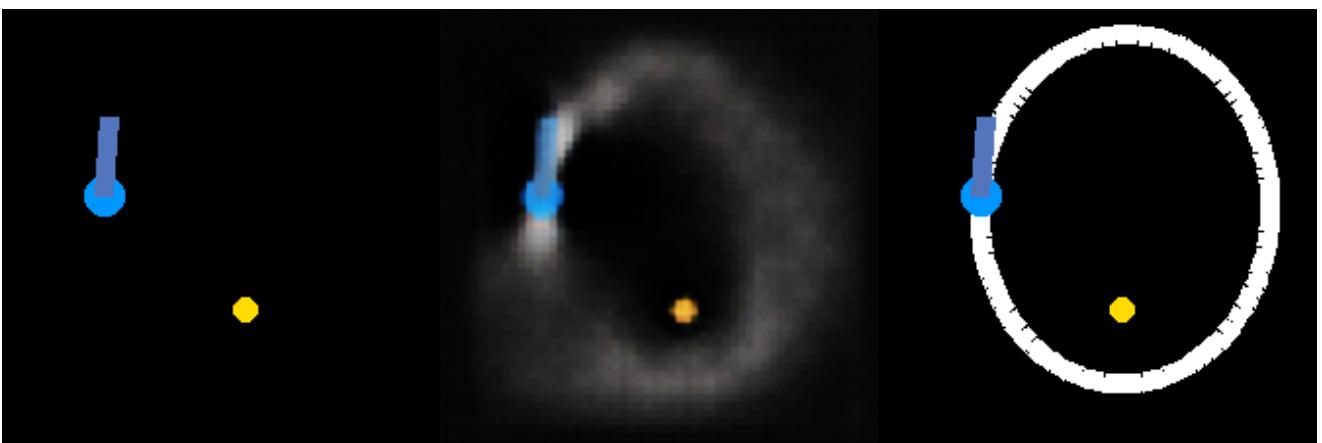
600step



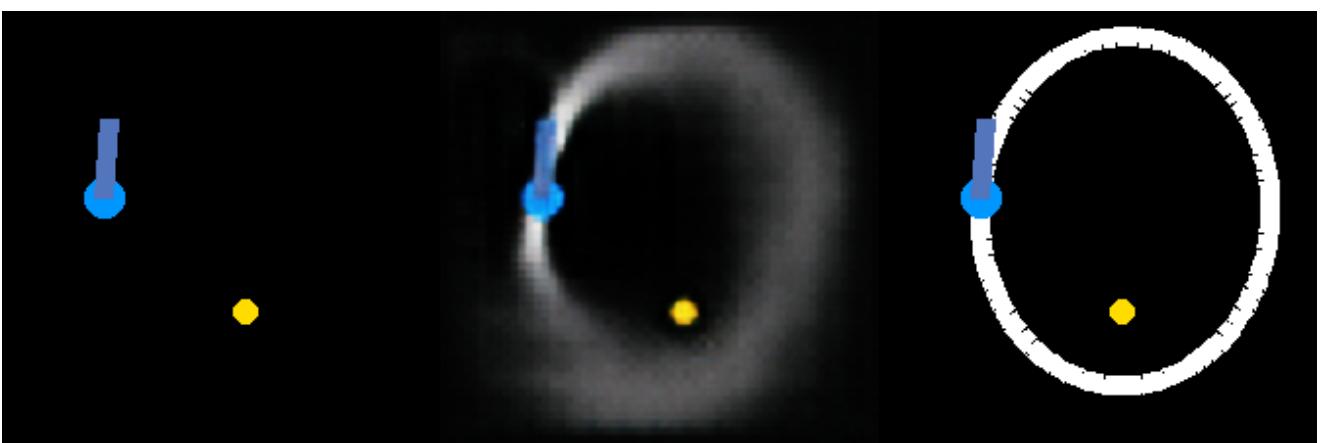
800step



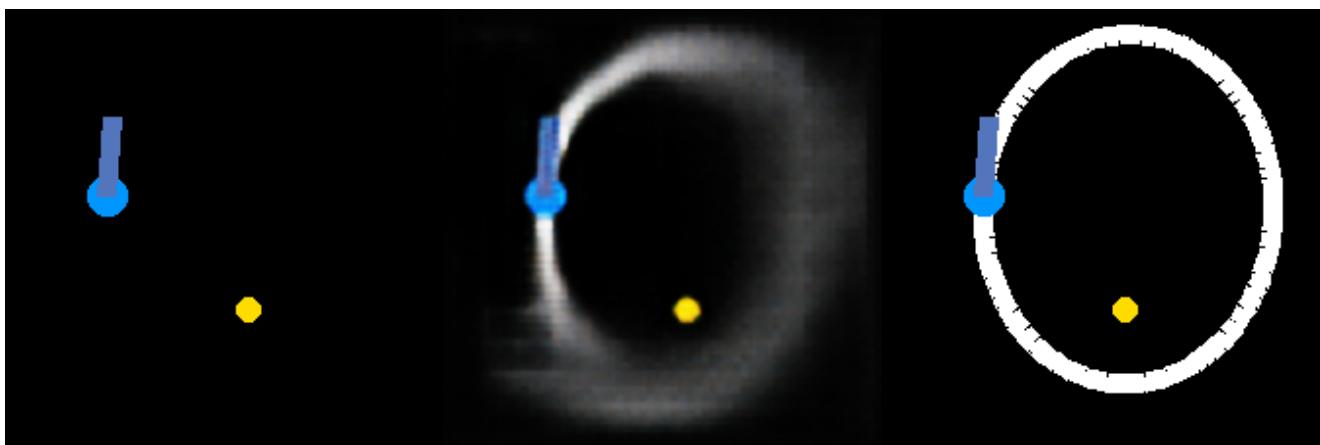
1000step



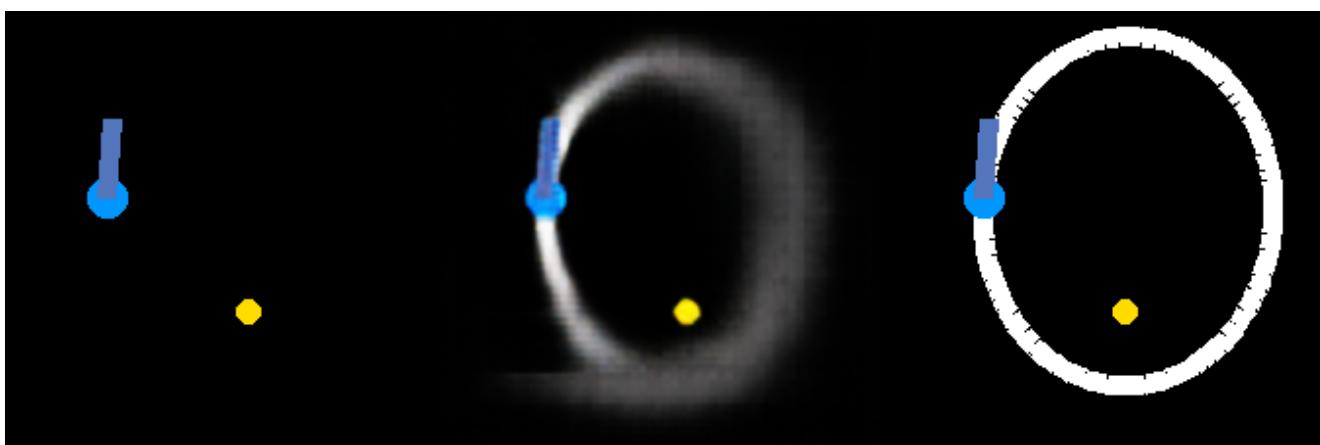
2000step



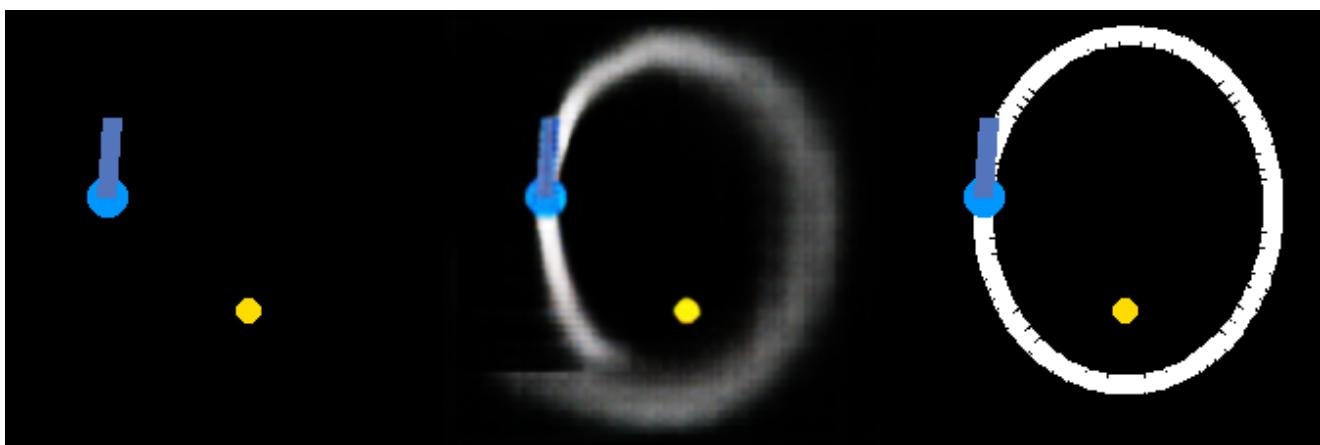
3000step



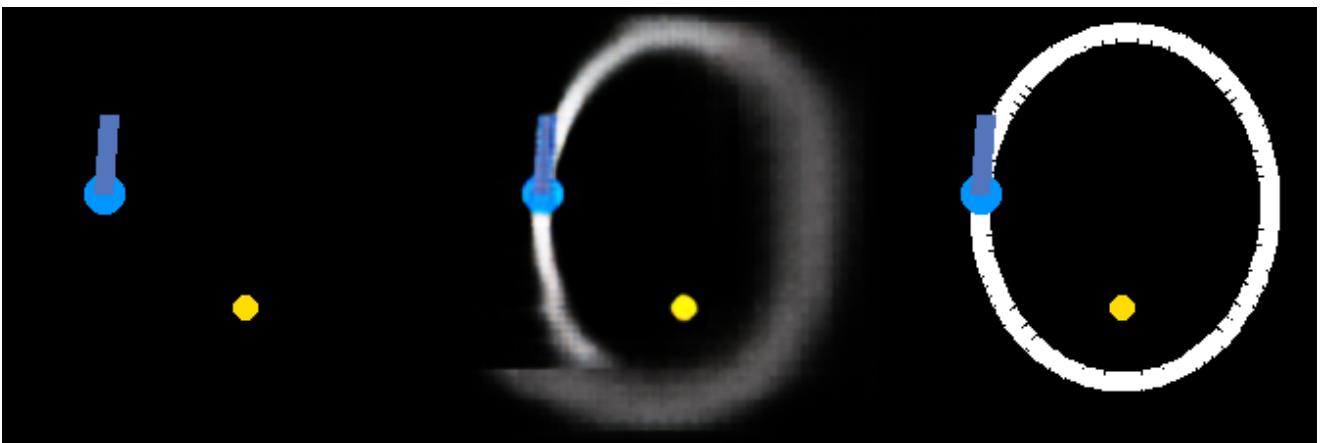
4000step



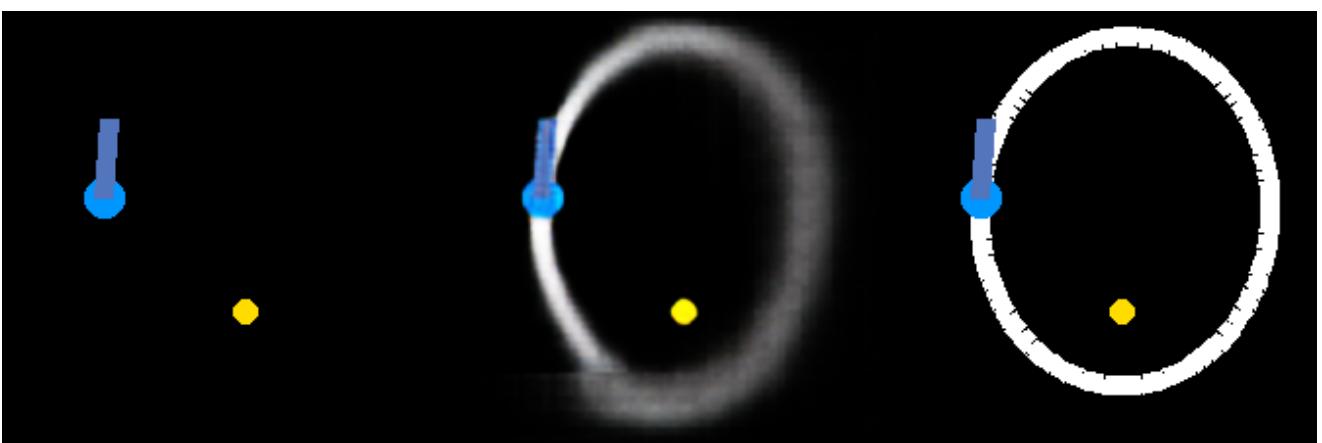
5000step



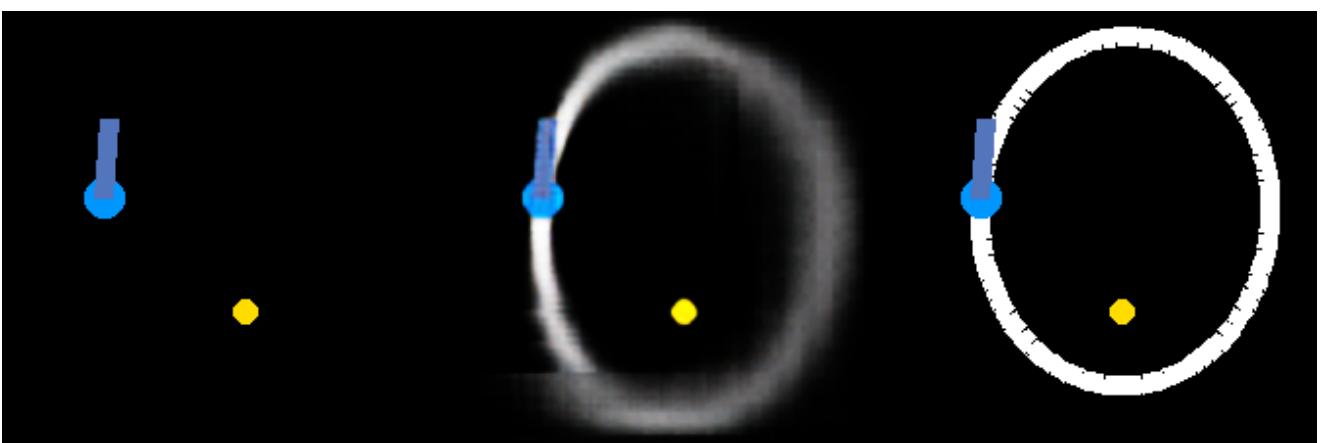
6000step



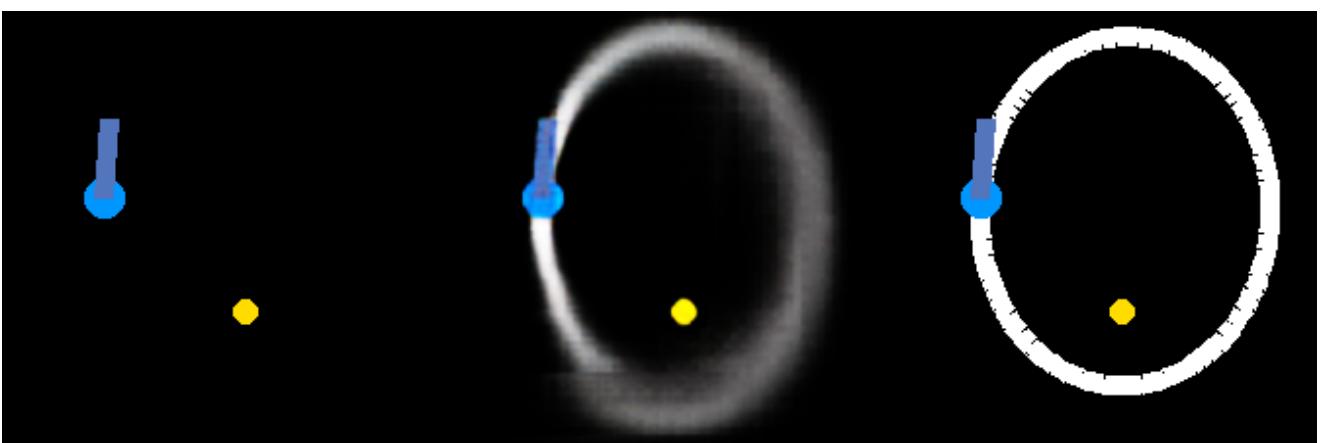
6000step



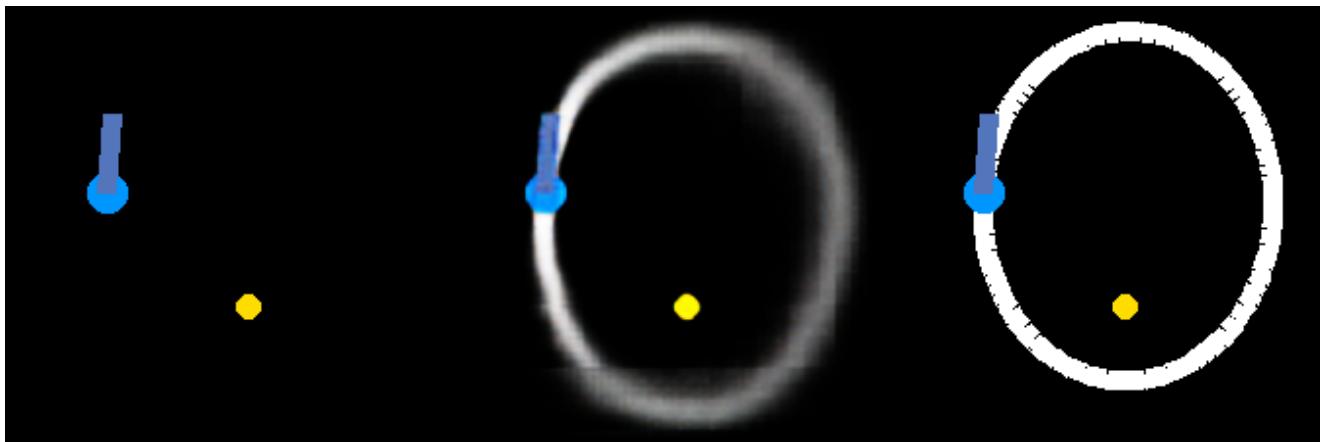
7000step



8000step

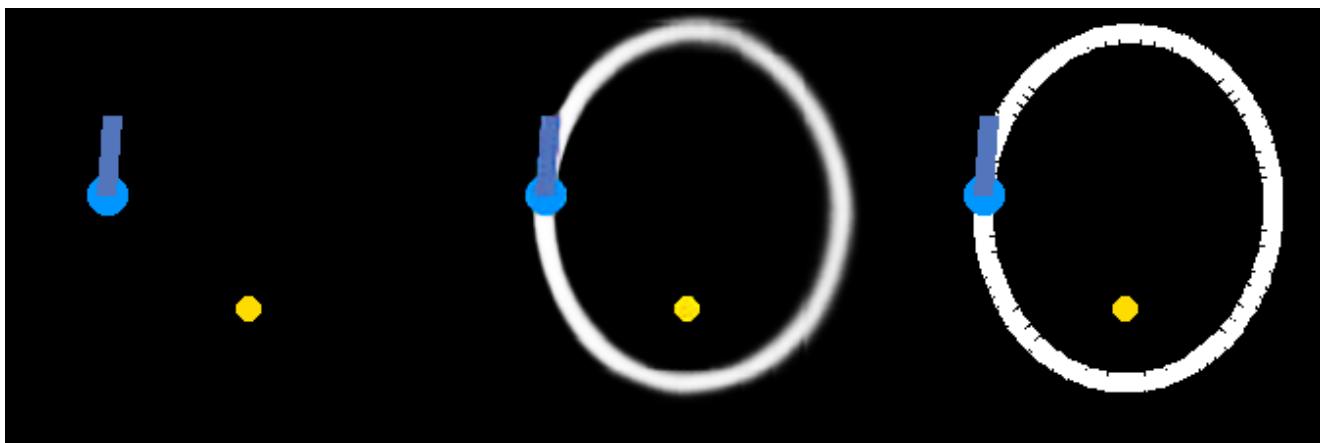


10000step



后面由于变化速度减缓，直接展示最后训练结果

156400step



4.7. arc-agi任务相关

本节将展示该范式在处理 [12,13] arc-ag-i-1/2任务上的能力。

我的实验方法介绍：

1.arc-ag-i-1/2数据集的模式都是给出非常少量的例子，3个左右，要求在这些例子上领悟游戏规则，在新的例子上完成泛化和解题。我试过用我的方法严格按照arc-ag-i-1/2的要求是无法完成的，会出现极其严重的过拟合。因此我采用了一种迂回的方法，我先手动观察一个任务的逻辑，通过与LLM（如gemini 2.5 pro）的交互式编程，来辅助编写和调试生成这些复杂任务的训练数据脚本，然后运行脚本生成大量数据集，一般在150000-200000左右，最后在这个数据集上训练并寻求泛化。

2.因为我的这种实验方法需要大量时间精力，包括人工观察任务的逻辑和生成及修改训练集生成脚本的时间，所以我只在16个任务上进行了尝试，8个arc-ag-i-1任务，8个arc-ag-i-2任务，按照随机挑选并且训练集生成脚本相对简单的原则。结果是相当好的，绝大部分问题很快就收敛了，少数问题需要长时间训练，也可以收敛。这种通用性是非常令人震惊的，如果考虑到我是在随机抽取16个问题。

3.采用图像推理模式，Swin Transformer+unet模型。

4.采用arc-ag-i-1/2数据集官网那种彩色图像的表示方式。

5.格点的横纵维度在整个数据集中保持一致。

6. 一般来说150000个数的数据集就足以完成训练，但限于时间精力，我没有尝试过最少多少就可以。

4.7.1. 水流模拟

任务描述：出自[ARC Prize - Puzzle #1ae2feb7](#)

输入格式：224*224图像

输出格式：224*224图像

loss：MSELoss

训练配置：4090显卡

训练集生成脚本：generate_arc_fluid_simulation.py

数据集：arc_fluid_final_dataset文件夹

训练代码：train_image2image.py

使用模型：Swin Transformer+unet

训练集大小：150000

输入空间大小估计：采用蒙特卡洛法，运行2000000次没有重复样本

训练集大小占输入空间大小的比例：如上，可以认为训练集大小占比输入空间极少

任务相关设计思想和讨论：

1. 为了进一步检验transformer纯粹的图像推理能力，我把尝试扩展到已有的arc-ag-1/2的数据集，但采用另一种迂回的方法，已在上面说明。

2. 一些任务参数：

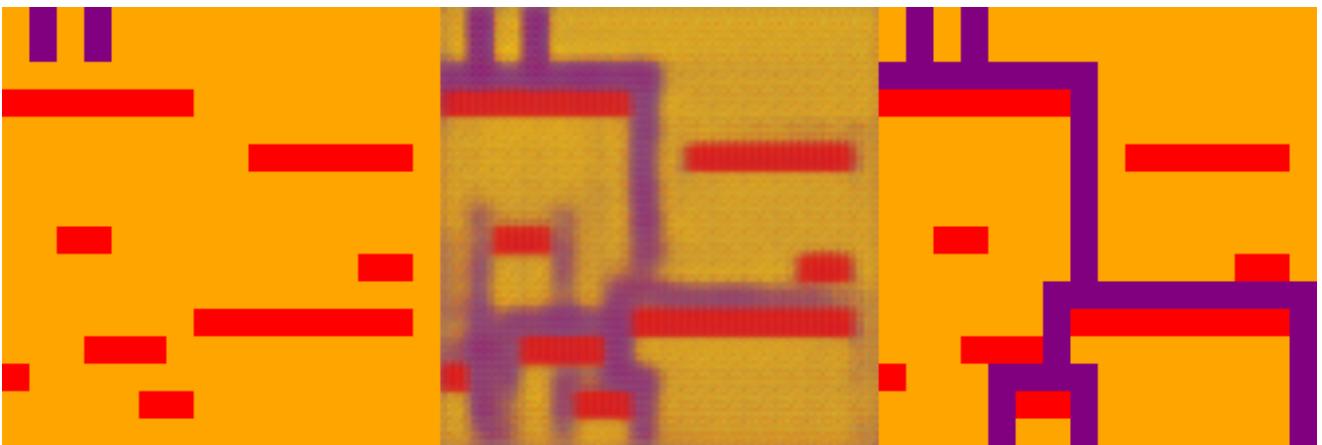
图像分辨率：224*224

实验结果与分析：最终的训练结果表明模型已经完全学会这个任务的规则，这表明模型具有arc-ag-1/2数据集所期待的推理能力，虽然需要大量数据集训练。同时，模型的学习速度非常快，提供了直接观察机器心智的形成过程。在这个任务上，可以看到模型一开始学会了简单的规则，它似乎猜测从红色挡板边缘流下紫色水流就是对的，在之后的过程中慢慢抛弃了这个过于简单的规则假设，最终学到了真正的任务规则。这证明了我的范式，结合“程序化数据生成”的方法论，是攻克小样本抽象推理难题的一条极具潜力的路径。而且这个任务本身的推理性质很大程度上排除了模型在进行朴素内插的可能。

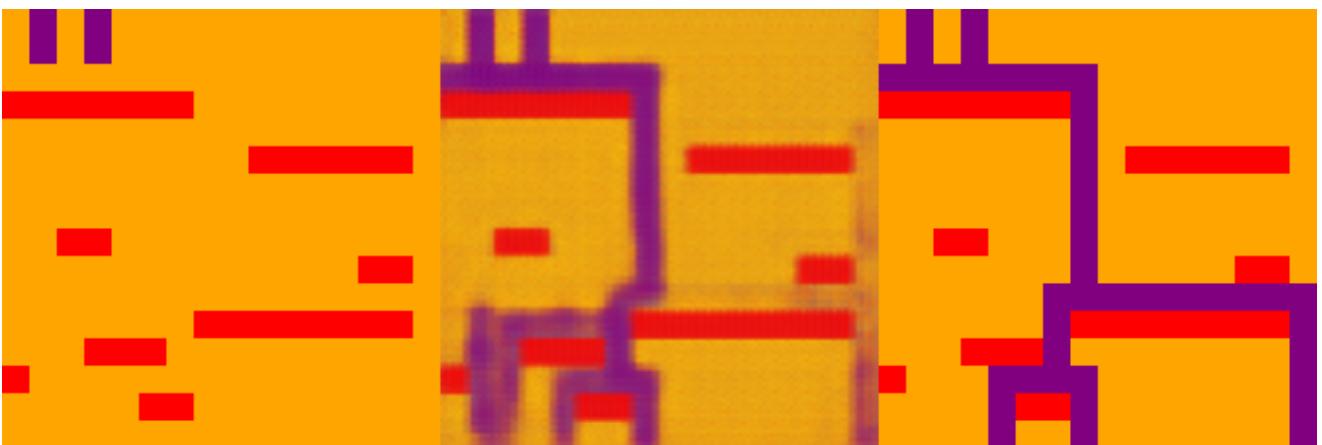
实验过程展示：

左边是输入图像，右边是ground truth，中间是生成图像。

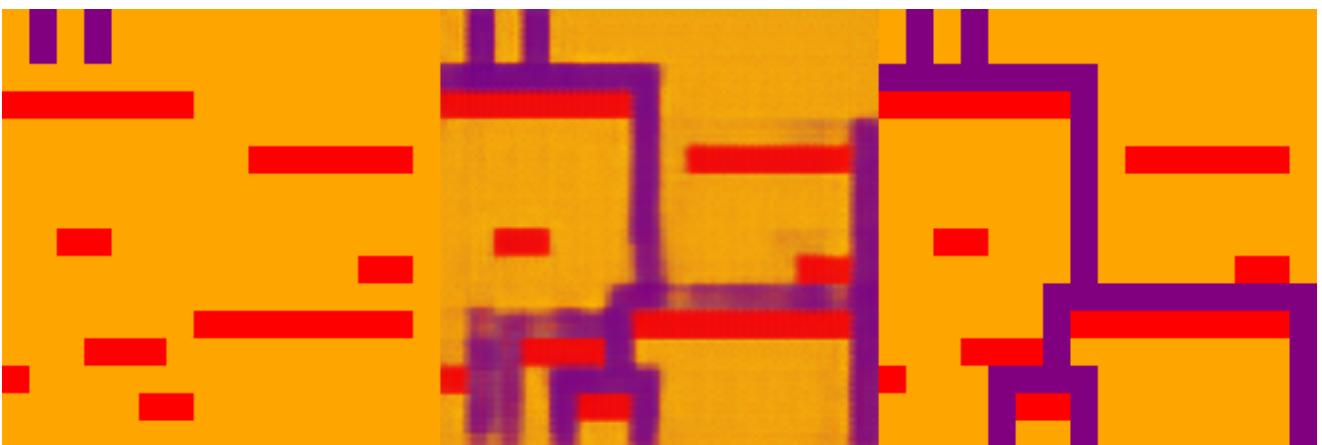
200step



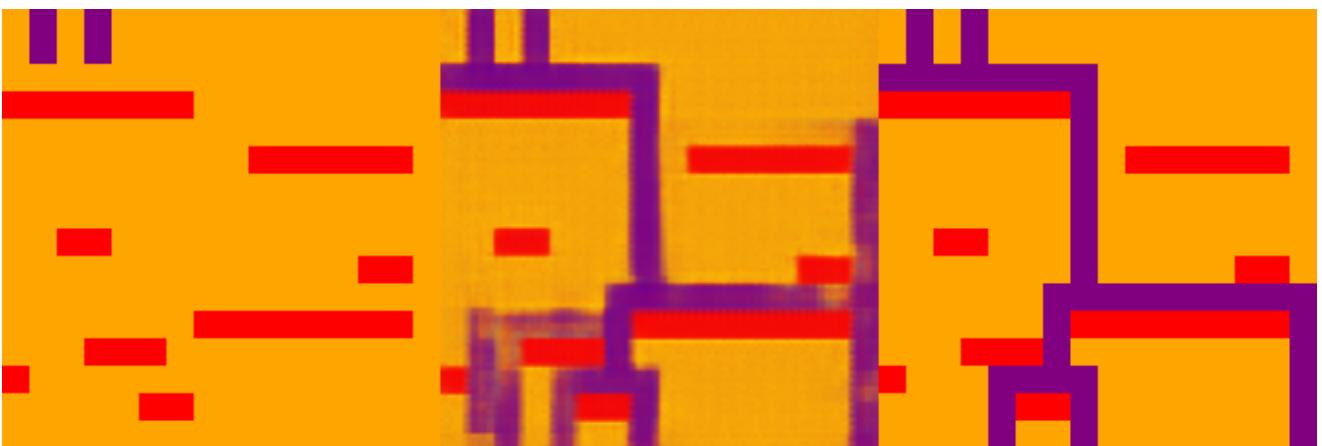
400step



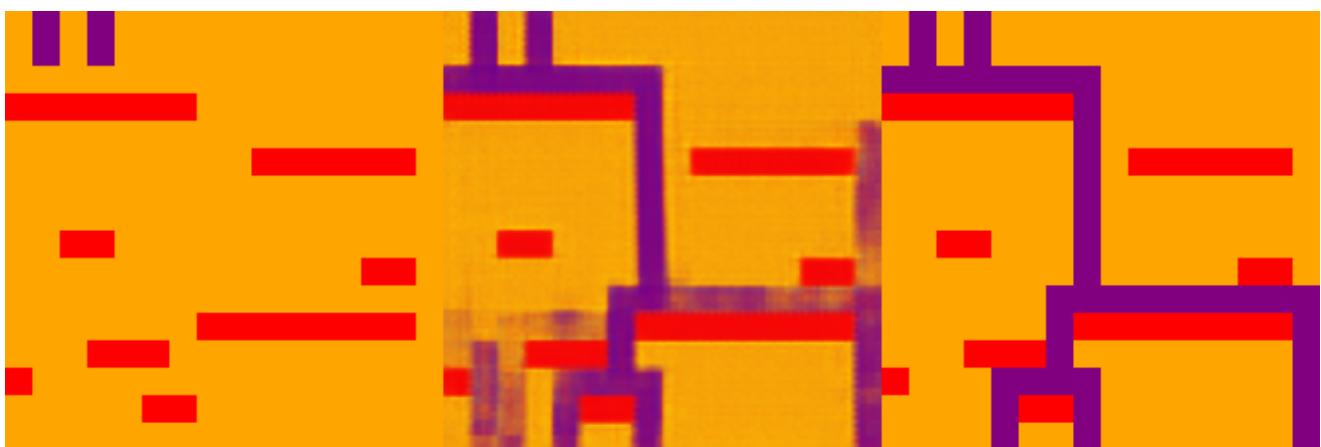
600step



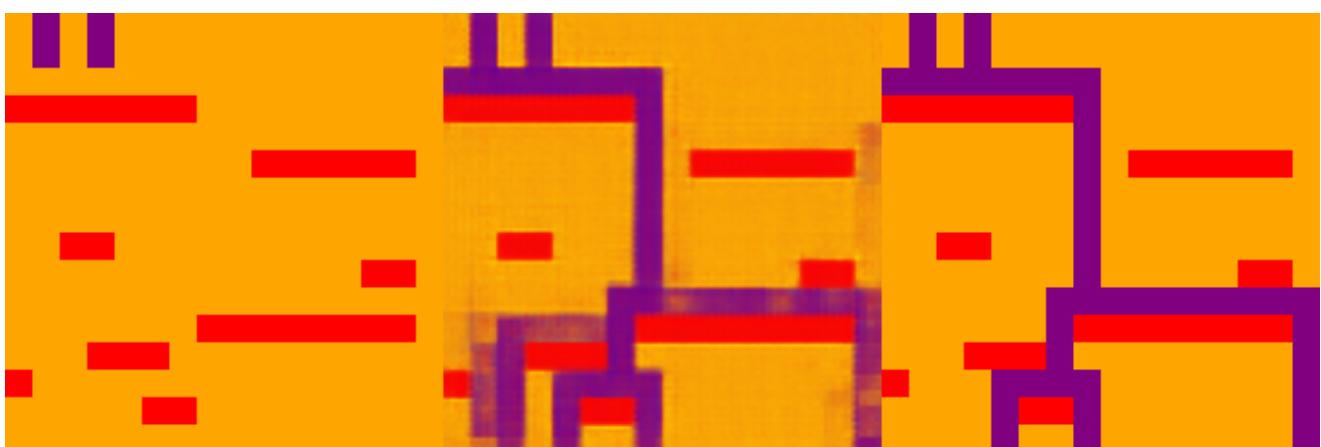
800step



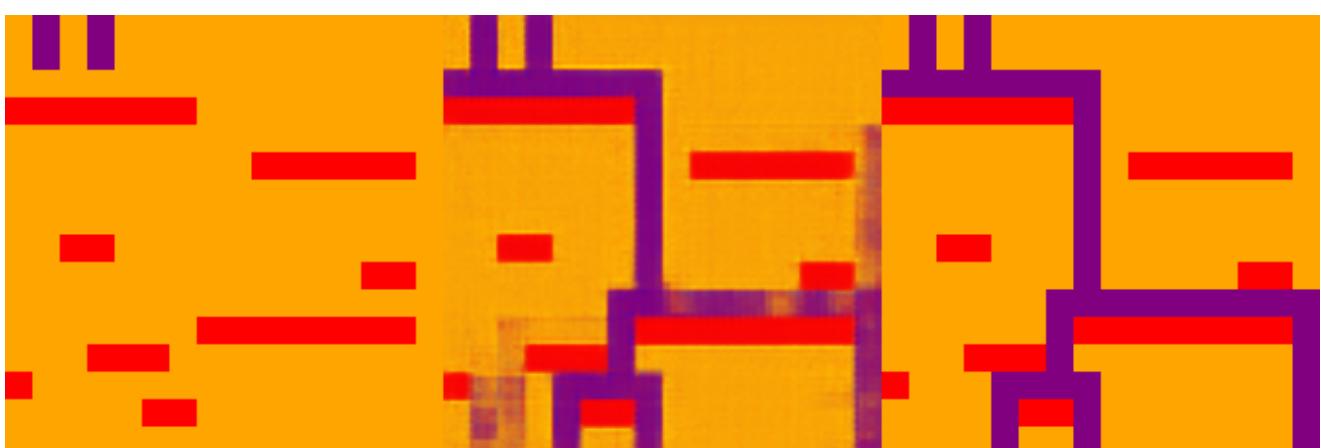
1000step



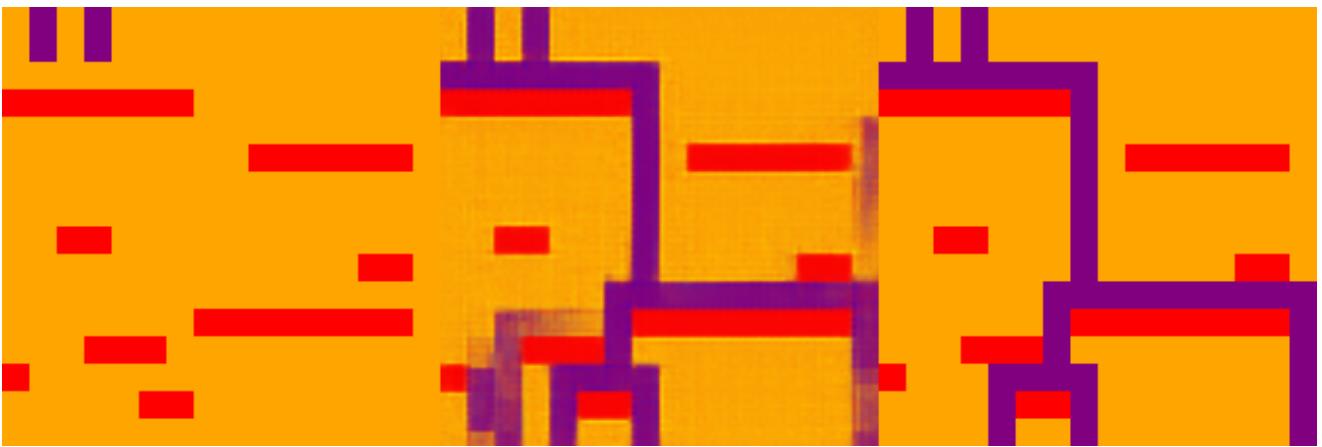
1200step



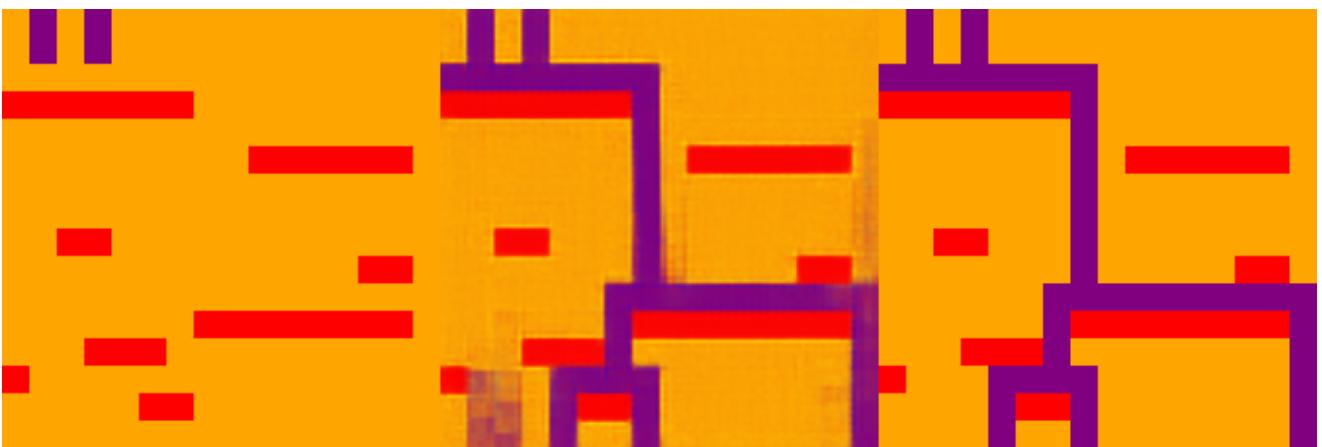
1400step



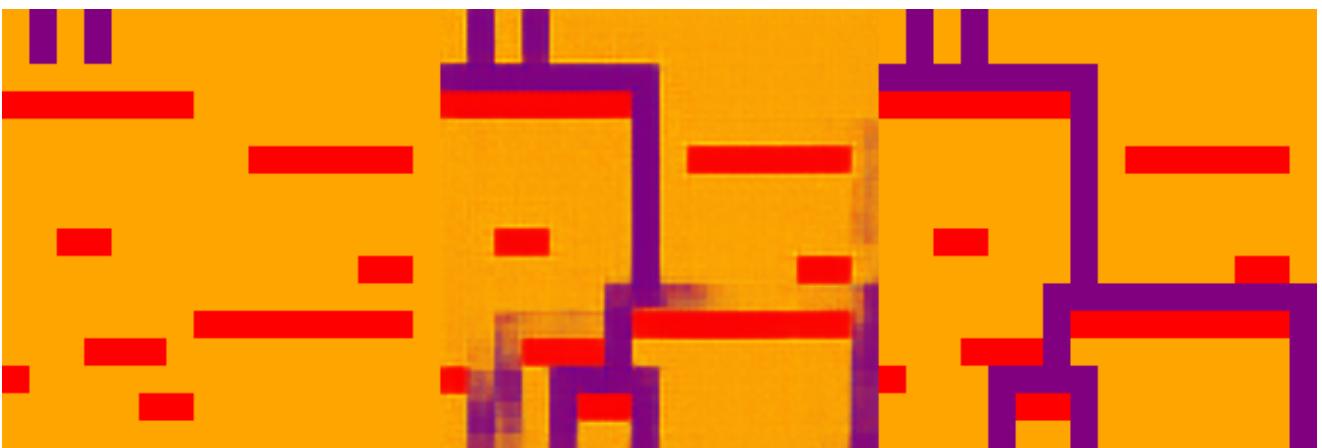
1600step



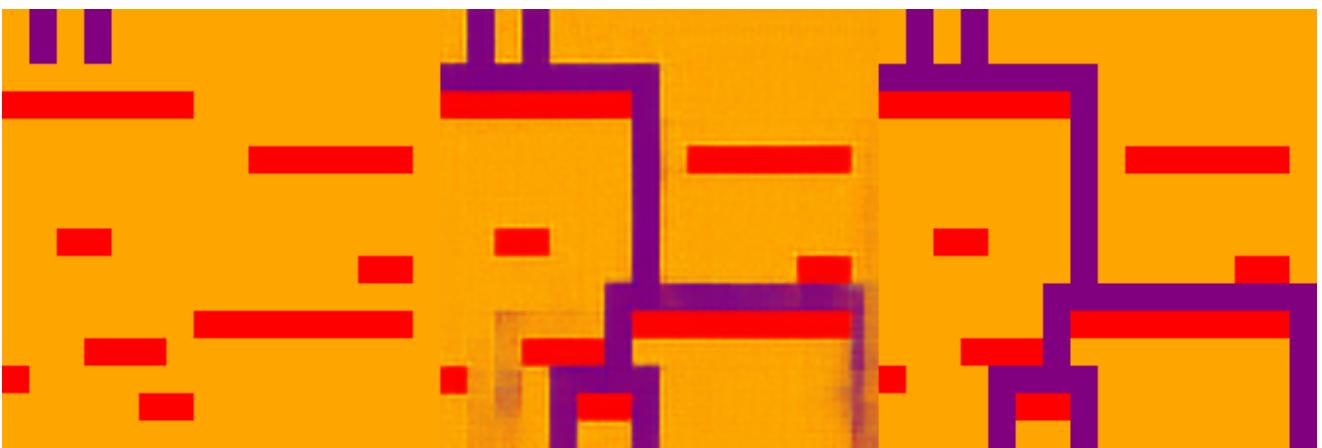
1800step



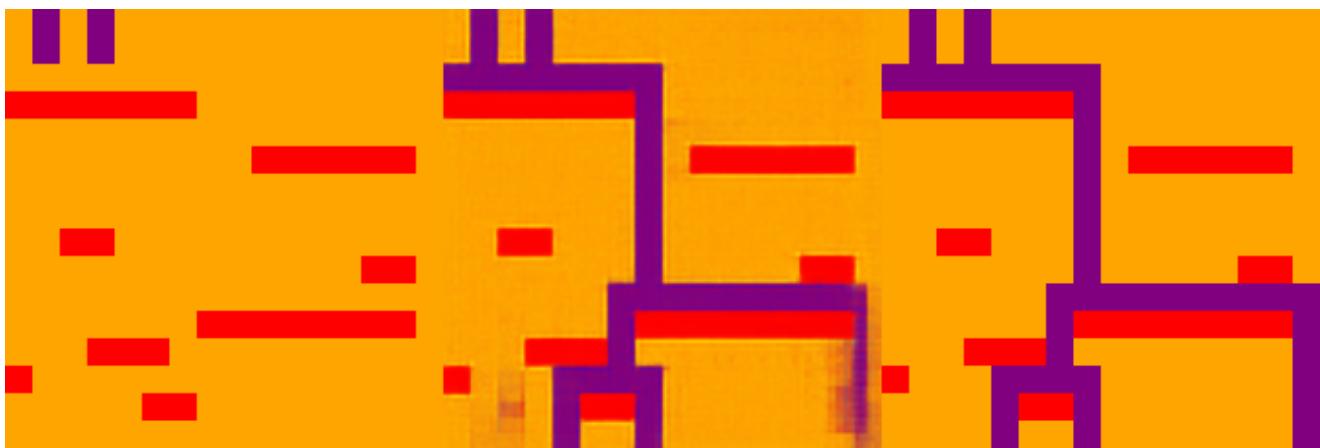
2000step



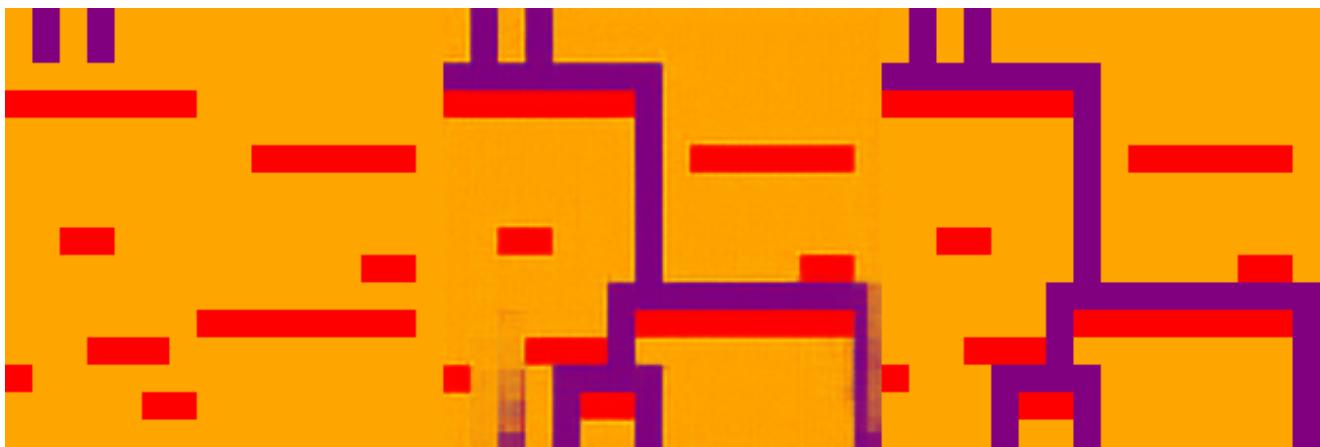
3000step



4000step

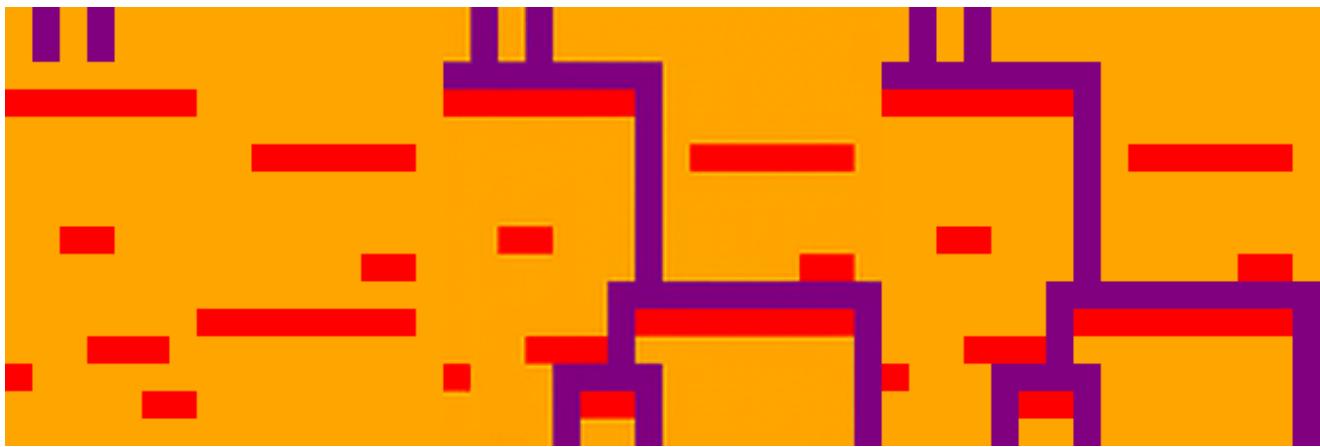


5000step



后面由于变化速度减缓，直接展示最后训练结果

39600step



4.7.2. 积木游戏

任务描述：出自[ARC Prize - Puzzle #1ae2feb7](#)

输入格式：224*224图像

输出格式：224*224图像

loss: MSELoss

训练配置：4090显卡

训练集生成脚本：generate_arc_jigsaw_puzzle_advanced.py

数据集：arc_jigsaw_puzzle_mine_dataset文件夹

训练代码：train_image2image.py

使用模型：Swin Transformer+unet

训练集大小：200000

输入空间大小估计：采用蒙特卡洛法，运行1000000次没有重复样本

训练集大小占输入空间大小的比例：如上，可以认为训练集大小占比输入空间极少

任务相关设计思想和讨论：

1.为了进一步检验transformer纯粹的图像推理能力，我把尝试扩展到已有的arc-ag-1/2的数据集，但采用另一种迂回的方法，已在上面说明。

2.一些任务参数：

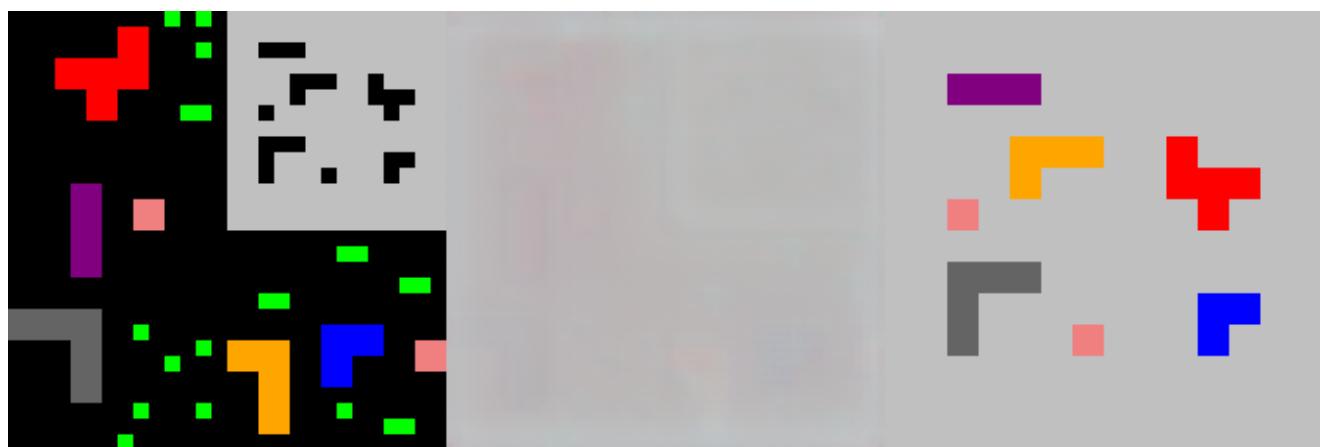
图像分辨率：224*224

实验结果与分析：最终的训练结果表明模型已经完全学会这个任务的规则，这表明模型具有arc-ag-1/2数据集所期待的推理能力，虽然需要大量数据集训练。同时，在这个任务上，模型需要较长时间学习，提供了直接观察机器心智的形成过程。在这个任务上，可以看到模型一开始学会背景色，可能也是因为一开始它占loss比重较大，而且相对容易学习，然后慢慢学会了需要填充的积木形状和位置，最后花费相当长时间学会了填充颜色的规则。这证明了我的范式，结合“程序化数据生成”的方法论，是攻克小样本抽象推理难题的一条极具潜力的路径。而且这个任务本身的推理性质很大程度上排除了模型在进行朴素内插的可能。

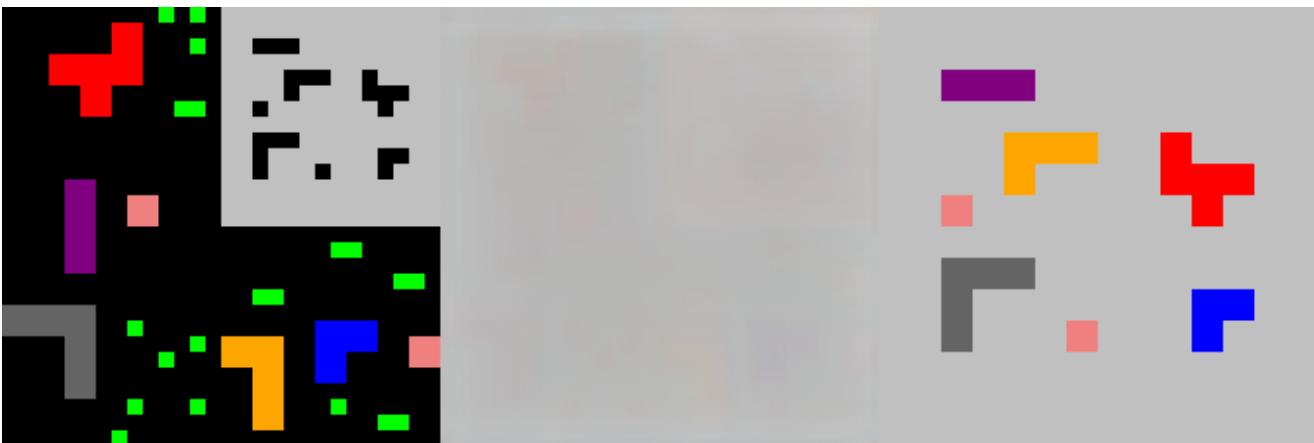
实验过程展示：

左边是输入图像，右边是ground truth，中间是生成图像

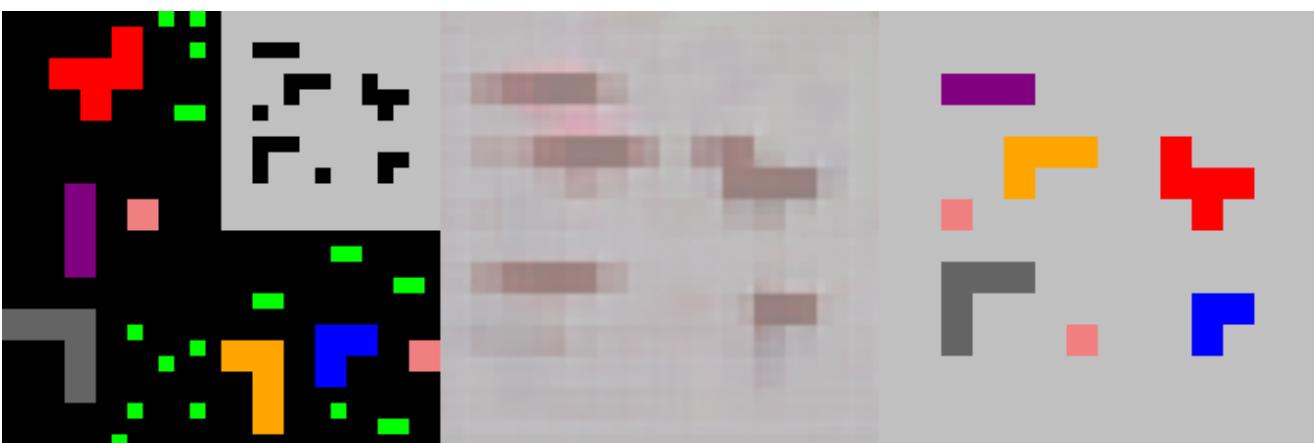
10000step



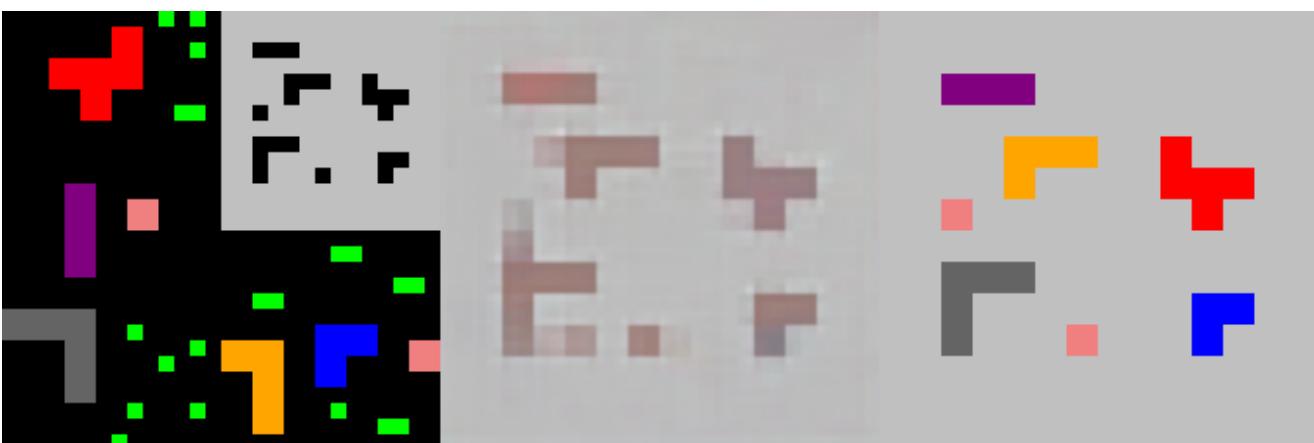
20000step



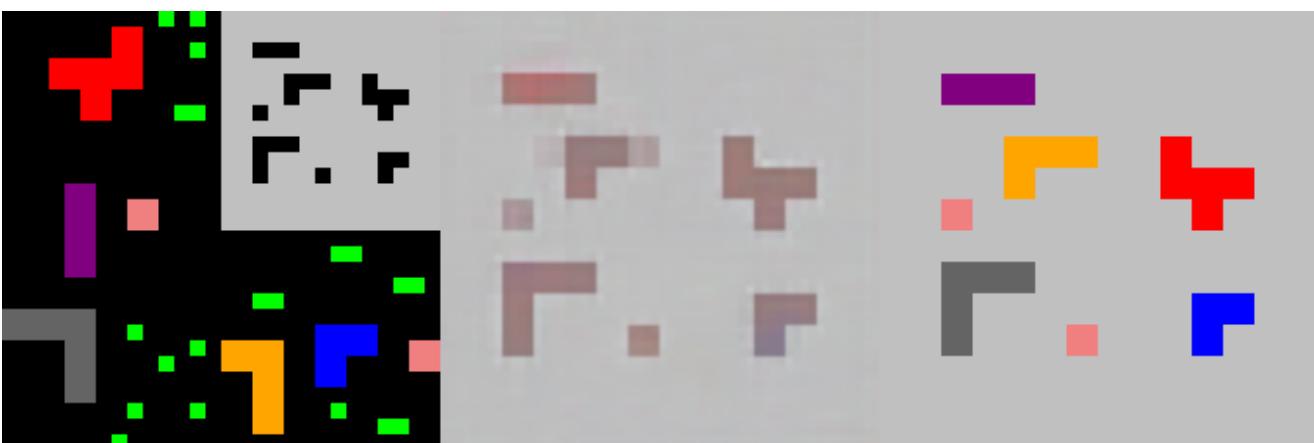
30000step



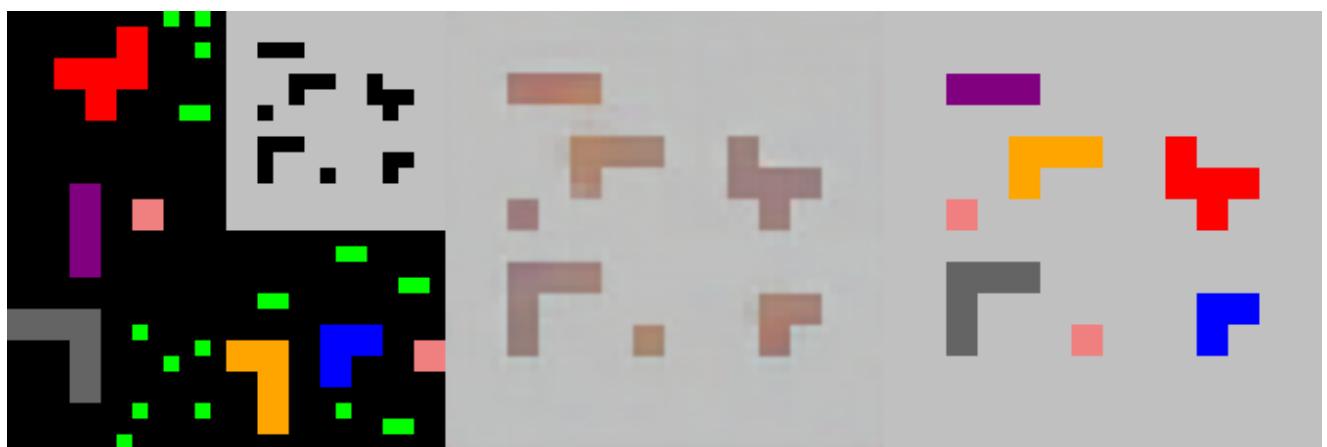
40000step



50000step



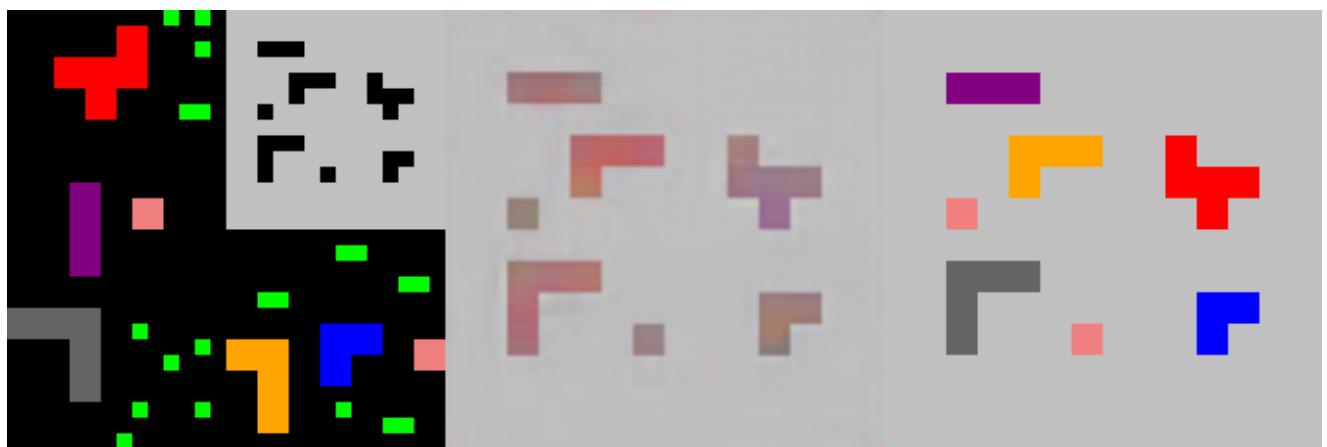
60000step



70000step



80000step



90000step



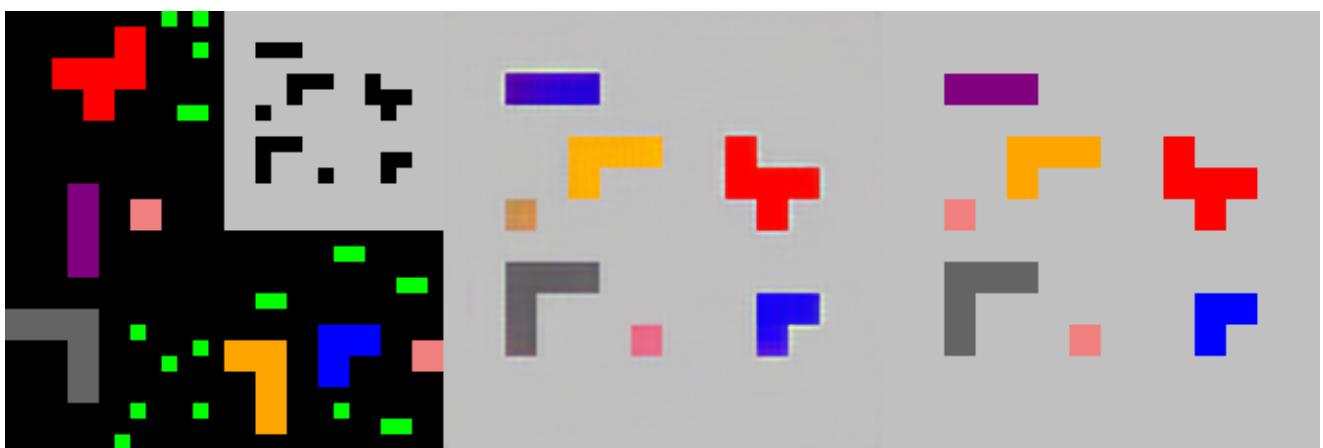
100000step



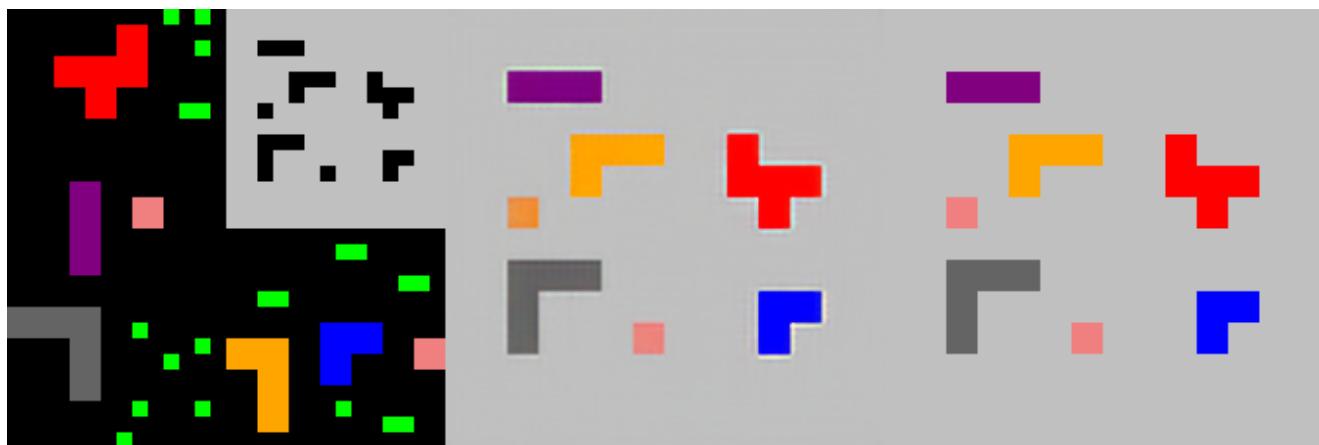
150000step



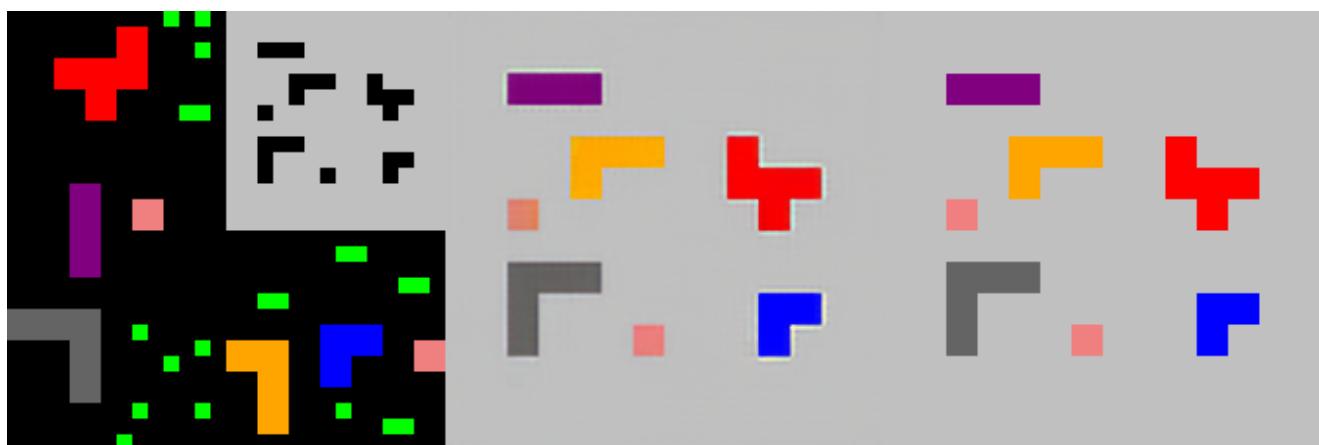
200000step



250000step

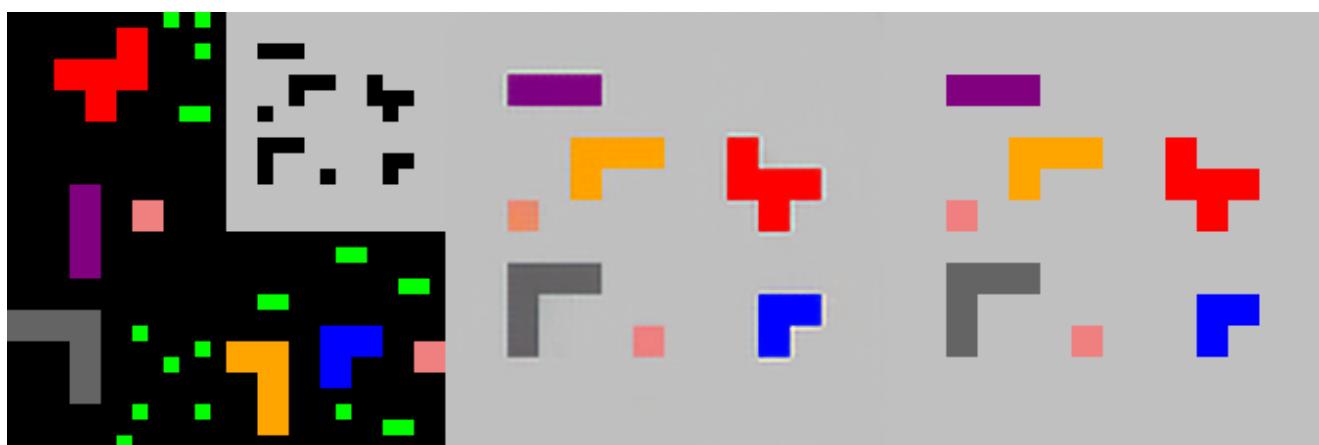


300000step



后面由于变化速度减缓，直接展示最后训练结果

530400step



4.8. 可解释性探索

本节将展示该范式输出推理过程的能力。

4.8.1. 编辑过程输出

任务描述：模型学习预测和输出两个0/1字符串之间的最短编辑过程

输入格式：30位的由“0”或者“1”组成的字符串，前面15位表示字符串A，后面15位表示字符串B

输出格式：450个标签的多标签二分类格式，等同于450个01字符串，每连续的30位分为一组，一组的前15位代表编辑过程中字符串的中间状态，后15位代表字符串的掩码，以区分子字符串编辑过程中长度小于15位的情况

loss：平均二元交叉熵损失

训练配置：4090显卡

训练集生成脚本：generate_edit_distance_explainable.py

数据集：edit_distance_path_unique_final.jsonl

训练代码：train_tiny_transformer.py

训练集大小：500000

输入空间大小估计： $2^{30} = 1073741824$

训练集大小占输入空间大小的比例：0.0466%

任务相关设计思想和讨论：

1. 这理论上来说并不算直接解释神经元权重或者某个模块的作用，而是一种在可解释标签指导下的可解释性。这种可解释性的操作方法是，直接把需要观察的内部过程加入标签，实践表明这种方法是实用的。比如有些问题，既然网络可以拟合答案，在很多情况，我都可以确信它必然也以某种方式了解内部的一些状态或中间过程，这个时候就可以通过这种方式暴露。至于更纯粹的神经网络的可解释性的原意，我认为这个范式可以做到更加干净的实验室环境来观察神经网络的权重变化，这个在后面会有讨论。

2. 一些任务细节：

- 通过脚本保证编辑过程中没有字符串长度超过15的时候，否则我的输出格式无法表示。
- 并且通过脚本保证最小编辑路径有且仅有一种，否则会大大提高网络收敛的难度。
- 编辑距离字符串选择限定在01两种，是为了降低任务难度，而不改变问题本质。

实验结果与分析：评估损失收敛至0.001404。这个成功表明，这个范式不仅能给出答案，更有潜力输出解题步骤，为解决神经网络的黑箱问题，提供了一个全新的、可操作的思路。

最终收敛结果：

--- Step 66000 ---

Train Loss: 0.001995

Eval Loss: 0.001404

第五幕：论可学习性与可解释性的边界

我总结了这个范式的特点

1. 输入输出格式要设计好，输入格式要尽量减轻transformer解码负担，输出尽量采用多类别分类对应交叉熵loss或者多标签二分类对应平均二元交叉熵loss。可以对应任务的特点自己设计输出格式的含义，只要保证整个数据集中的格式一致就可以，例如如果输出是一个数，可以用多标签二分类表示一个对应位数的二进制数。

2. 关于学习难度相关

之前在做符号规则学习任务的时候观察到一个现象，即任务的结构也很重要，看起来似乎是顺向符号规则任务才容易学，根据输出逆向求解输入则不容易学，除非这个逆向其实也是某个顺向符号规则的形式。现在看来可能是因为这种“逆向求解”任务更类似算法模拟问题，所以学习难度更大。

观察到的另一个现象是，带有分支条件判断的任务可能会加剧学习难度，比如某一个指示位指示用完全不同的规则计算后面的操作数。现在想想，分支判断其实也是规则本身，只不过会加剧符号规则的复杂程度，让它变得更难学习。

另外，输入中可以留有控制任务规则的位置，即输入=任务规则+操作数。这个设想已经在元胞自动机相关实验中得到成功验证。

另外，观察到的另一个现象是，串行混合过多简单逻辑的任务也很难学习，假如一个符号变换规则串行组合不同简单符号规则，而每一个规则本身学习起来都很简单，它仍然可能组合成一个极其难以学习的任务。一个简单的例子是实验部分的接雨水实验，通过把最后的雨水量加和步骤取消，对问题进行解耦，才使得那个任务得以快速收敛。

顺向符号规则学习有难易程度之分，比如元胞自动机1维30位，不同rule的学习难度不同，应用不同层数rule的学习难度也不同，基本上层数越多学习越难，还有一些基本不可学习的任务，比如rsa加密，虽然是一个顺向的简单数学计算，但确实无法学习。

所以说规则难度影响loss下降速度。关于这个，举一个具体例子来说，元胞自动机，rule 110，演化层数，至少在较少的时候，严格遵循一个层数越多，训练越困难，模型收敛越慢的规律。这让人联想到wolfram提出的 [16] 计算不可约性，他也确实是拿元胞自动机的演化来举例子的。

另外，训练过程本身就是一种计算不可约性，训练/雕刻的过程是不可省略的。而且越简单的问题，训练越快，越难的问题，训练越慢，如果不考虑这个问题的本质是否对神经网络友好，输入输出格式是否对神经网络友好的话，控制好不同问题的输入的规模大致一致，训练过程中loss下降的速度快慢甚至可以用来估计这个问题的可学习性或者计算不可约性。比方说我曾经试图训练rsa加密，loss就完全不下降，始终没有学习到任何东西。这里还是拿元胞自动机举例，输入和输出都是30位01字符串，那么rule的类型和演化层数就可以完全决定输入到输出的映射。但这只是输入空间到输出空间映射方式的一种。为什么这一种就会比——演化层数更多的元胞自动机映射规则更容易学习？也显然比完全随机permutation的映射会更加容易学习？——完全随机permutation的映射显然是无法学习的，因为不包含任何规律。这是非常有意思的问题，但对它的探讨超出我的能力范围。另外，rsa算法的无法学习证实了密码学家创造的简单算法在这个意义上是非常成功的。

所以，可以通过这个，有可能通过实际训练任务的方法，这种类monte carlo方法，测定任务难度，可学习性/计算不可约性的量化衡量。这在数学上可能有潜力，为什么有的规则就被认定为复杂，有的就被认定为简单？以及这是否会形成一个新的理论？

另外关于模型容量，按直觉猜测应该是模型容量更大/更适合这个任务，会让训练更快。因此，可能是模型容量和任务难度结合会决定一个训练的loss收敛速度和收敛过程。

3.关于如果训练集中的变换规则不一致时会出现的这样的情况。即使是简单规则，如果训练集中不同的数据应用不同的规则，也很难学习，因为在该应用什么规则没有任何指示，模型只能

按照降低loss的原则去取得某种平衡。如果混合了很多规则，但又无法被识别该用哪个简单规则，这个时候训练loss和eval loss会在一个高点震荡，因为不同规则在拉扯到一个均衡点。这个loss的值的高低将会取决于训练集中规则的纯度。

另外，mod 3是一个令人非常意外的无法学习的规则。它非常简单，但loss又始终停留在高位，有可能和那个规则混合的现象有关，但是它至少在人类的视角直观看起来又不是混合规则。我做了一个实验，输入是20位2进制数，由20位01字符串表示，计算它的mod 3结果，由2位2进制数表示，loss是mean bce loss。现在有趣的结果来了，我生成了四个数据集，第一个数据集是不对输入有限制，第二个数据集是限制mod 3不能为0，第三个数据集是限制mod 3不能为1，第四个数据集是限制mod 3不能为2。训练的结果是，他们分别eval loss稳定在下面的结果：

不做限制的数据集——0.636

限制mod 3不能为0——0.693

限制mod 3不能为1——0.347

限制mod 3不能为2——0.347

分析这个结果是很有意思的，我先给出一个事实，就是假如模型从数据中学不到任何东西的话，对于一个bce loss，假如标签为1的概率为p，那么模型会把loss降到

$$-p \ln p - (1-p) \ln(1-p)$$

。之后我就会看到这些loss结果证明模型没有从数据中学到任何东西，即mod 3是无法学习的。原因如下：

对于不做限制的数据集，输出的2位二进制数，高位为1的概率是 $1/3$ ，低位为1的概率也是 $1/3$ ，因此最佳

$$\text{loss} = -\frac{1}{3} \ln \left(\frac{1}{3} \right) - \frac{2}{3} \ln \left(\frac{2}{3} \right) = 0.6365,$$

与实验结果非常接近。

对于限制mod 3不能为0的数据集，输出的2位二进制数，高位为1的概率是 $1/2$ ，低位为1的概率也是 $1/2$ ，因此最佳

$$\text{loss} = -\frac{1}{2} \ln \left(\frac{1}{2} \right) - \frac{1}{2} \ln \left(\frac{1}{2} \right) = 0.693,$$

与实验结果非常接近。

对于限制mod 3不能为1的数据集，输出的2位二进制数，高位为1的概率是 $1/2$ ，低位为1的概率是0，因此最佳

$$\text{loss} = -\frac{1}{2} \ln \left(\frac{1}{2} \right) = 0.347,$$

与实验结果非常接近。

对于限制mod 3不能为2的数据集，输出的2位二进制数，高位为1的概率是0，低位为1的概率是 $1/2$ ，因此最佳

$$\text{loss} = -\frac{1}{2} \ln \left(\frac{1}{2} \right) = 0.347,$$

与实验结果非常接近。

因此，可以认为至少在这种实验设置下，mod 3是无法学习的，而且不是因为拟合能力不足。

在训练汉诺塔实验中有类似现象，具体表现为，假如我只训练一个固定盘子数量的汉诺塔问题，从初始态到终止态的最佳路径上的所有状态，训练这些状态的最佳action，会发现收敛非常迅速。但当我试图把状态扩展到固定盘子数量的整个汉诺塔状态空间时，会发现loss停留在一个较高的位置完全不再下降，而且很可能并非拟合能力的问题。注意观察汉诺塔的整个状态空间和mod 3问题的相似性，会发现mod 3在2进制格式下是有点类似递归或分形性质的，汉诺塔就更加是天生具有递归或分形性质。那这是因为输入格式或者输出格式的设计原因导致无法学习，还是因为问题本身？是否这个范式无法把涉及递归或分形的规则有效识别出来？我倾向于后者。那么，为什么只在汉诺塔的初始状态到终止状态的最佳路径上训练，就可以快速收敛呢？答案应该是因为，在这个最佳路径上，有一个非递归的识别规则。

那么这两个和递归或分形相关的问题，是否和最开始提到的混合规则有关系？本文认为可能是有关的，因为当模型无法识别递归规则时，它可能会认为是某种复杂的混合规则形式，而且它无法识别哪个输入对应哪个规则。

4.在这个范式中，很多时候数据可以由程序无限廉价生成。理论上我需要使得训练集和整个输入空间同分布，那么最简单的方法就是在输入空间随机采样。但这里会有另外一个问题，会不会存在另一种训练集的构造方式，使得模型的学习效果更好？比如假如规则的某一个分支状况出现地较少，随机采样很难覆盖足够的模式。我的设想是也许可以有更加精妙的数据集生成方式，更好地覆盖这个问题的各种模式。因为根据实践经验来看，只要有足够的模式覆盖，训练集就可以达到效果，具有抗过拟合的能力并使得满足模型学习的需求，这也是训练集的总量并不需要非常大的原因。如果从这个角度分析，过去模式识别的深度学习范式，需要大量训练集的原因就比较清楚了，因为那些任务没有精确的变换规则，需要更多的数据去覆盖输入空间的各种模式。

这个范式不追求ood泛化。换句话说，由于训练数据占整个输入空间的比例非常小，除掉训练数据的整个剩下的极大的输入空间，收敛后的模型都可以精确泛化，这个可以类比过去范式的泛化。

5. 可控ai和可解释性

关于可解释性，在上面解释过，这理论上来说并不算直接解释神经元权重或者某个模块的作用，而是一种在可解释标签指导下的可解释性。这种可解释性的操作方法是，直接把需要观察的内部过程加入标签，实践表面这种方法是实用的。比如有些问题，既然网络可以拟合答案，在很多情况下，我都可以确信它必然也以某种方式了解内部的一些状态或中间过程，这个时候就可以通过这种方式暴露。

关于更可控的ai，由于这个范式的特点本身，非自回归的transformer模型，拟合精确规则，从理论上是不会出现类似LLM那种幻觉或者cv中的对抗样本的。实践中也确实没有发现类似的幻觉。原因可能是，由于这个范式是基于规则的任务，内在规则极其精确，本质上不是光滑的，无法通过内插解决问题，因此模型只能学习规则，所以天然抗幻觉。

另外一个相关的发现是，训练过程中batchsize不能过小，batchsize如果过小，学习会慢很多，推测是因为这个范式的任务是精确规则，每一个样本的梯度方向相差会更大，所以需要batchsize达到一定规模去平均梯度。

6.在训练过程中的发现，训练loss和验证loss同步下降，而且验证loss基本始终低于训练loss。而且这个范式非常抗过拟合，除非输入空间特别大，问题特别复杂，以至于训练集大小没有满足覆盖问题所有模式的要求。

7.这个范式学习的是符号规则，它看到0和1的距离与0和某个emoji的距离是一样的，都是符号的占位符。“结构”才是真正的“语言”：它最终学会的，不是0代表0，也不是1代表1。它学会的是一个更抽象的东西——无论这些符号长什么样，它们背后都对应着一个可以进行加法运算的、具有特定群论和环论性质的数学结构。它与这个抽象结构进行对话，而非与具体的符号。这个已经在前面的n进制加法实验中得到证实。

8.关于这个范式整体的哲学含义

- 数据，要求覆盖问题的各种模式，并要求和整个输入空间相同（是否一定要和输入空间同分布，有待商榷，但至少这是一个最方便的做法。），不要求绝对数量，当然了数据越多越好是肯定的。数据，就好像是给神经网络创作参照的样品或者模特。为什么数据越多越好，为什么类似于monte carlo算法的思想，因为不需要显式逻辑的计算，只需要数据和训练，数据越多，给神经网络提供的参照样品或者模特就越清晰越具体。训练越多，神经网络拟合的就越好。从数据编程的角度来看，数据集的规模和质量决定模型效果的关键。
- 范式整体的比喻，就好像是，神经网络要在一个大理石上雕刻出类似数据的输入输出模式的样子，数据是模特，神经网络是雕塑师。用算力/训练，就好像在大理石上雕刻，是用连续性去逼近一个确定性的符号变换规则或者拟合一个未知但确定性的算法。而训练不足，loss尚未下降到极致，就好像雕刻尚未完成，如果问题过难，超出了这个神经网络的能力，就好像这个雕塑需要的能力超出了雕塑师的能力。而就算训练不足或者问题过难，神经网络也进行了拟合，也有效地压缩了输出空间的不确定性，就好像一个雕塑尚未完工，也比原来的大理石更像模特。

而且本文认为这个范式最有意思的一点是，可以连续性地逼近一个确定性的符号规则或者算法，本文认为这甚至在数学上也是一个很有意思的过程。而且本文认为我的范式有潜力在人们暂时没有找到通解方法的问题上，压缩问题的输出的不确定性，尽管可能无法给出精确的算法过程。这是在用神经学习一个符号或者算法任务，但是用它独特的方式，绝对不是以我人类以为的一步步理所当然的步骤，而是耦合的loss梯度下降，它只是在想办法降低loss而已。例如在一个算法问题上，假如想要在已经训练好的权重里，解释哪一部分是在执行算法步骤的哪一步，我认为是不可能的。它不知道输入符号的位置和含义，它只是在做梯度下降。

我现在感觉这个范式和Demis Hassabis所说的提倡压缩结构直觉和Ilya坚持的压缩即智能的信念是一致的。[1] AlexNet证明了之前手工设计的特征是不够好的，用联结主义直接学习图像特征，会得到更好的结果。而这个范式证明了，即使对于精确的符号规则，这个路线在某种程度上也是正确的。而且，正如前面所说，即使在无法完全收敛的任务上，模型也往往可以压缩答案的不确定性。

至于为什么这种联结主义的压缩尝试可能是有效的。我的感觉是，Demis Hassabis提到的，世界的复杂度正在超过人类语言与逻辑的承载极限，这种解释可能有局限性。我联想到Bob

dylan的歌曲Things Have Changed中的一句歌词，All the truth in the world add up to one big lie。但这并不意味着逻辑的失效，我的观点是，一个很长的逻辑链条的每一个命题都是有自己独特的语境和上下文的，而很多时候人们把他们组合为逻辑链条时会在某种程度上忽视这些语境和上下文，导致最终的逻辑链条的失真。现实生活中也经常可见此类场景，一个人在讲他的思考，尽管他说的每句话似乎都是正确的无可辩驳的，但最终却导向了一个荒谬的结果。这时候联结主义就派上用场，直接对接真实数据和现实验证，重新校准自身的直觉和逻辑。这就像符号规则从联结主义中浮现，人类的思考也是先积累观察和经验，再总结规律的，先归纳规则，才有演绎，演绎是无法替代归纳的，正如前面所说，如果不严谨地对待演绎，演绎会变得非常脆弱，这时仍然需要诉诸联结主义对应的观察和经验以及现实验证。

而为什么本文认为单纯的结构压缩可能不够完整，是因为我联想到混合规则的尝试，直接压缩是不够好的。也就是前面提到的，混合多个简单规则，会让学习这个任务的难度飞快提升。而如果对问题的结构有洞察，就可以通过解耦来降低问题的难度。也许这是逻辑认知的作用，人无法仅仅通过直觉应对这个世界。

9.通过这个范式来探索黑箱应该会更简单，首先，这个范式对于可训练的问题，loss可以降到非常接近于0。这种情况，第一它确实是梯度下降黑箱学到的参数，第二，和其他的图像识别，nlp不同，符号规则被学习可以更加被有迹可循地研究，对同一个模型分别训练不同的可训练问题，然后研究训练过程中的神经网络的权重变化或者验证输出的变化等等，比之前的模式识别的图像和nlp、音频的范式更容易在实验室的无菌环境研究。

而且，更有利于这一点的是，人工设计需要的任务规则，并且生成数据集是非常简单的。

同时，图像推理相关的实验可以可视化机器心智的形成过程。例如，我在三角形画内切圆的任务中，训练隔一段时间step就会保存一个当前eval image，就可以很直观地看到学习过程的进展，以及模型可能是以什么方式学习相关任务的。

10.我这个范式，如果以后可以发挥它真正的潜力，完全可以在不知道一些系统的工作原理的情况下，也不需要编程，只需要训练神经网络，就可以根据新的输入，预测可靠的输出。有些类似于Andrej Karpathy所说的 [22] 软件2.0的模式。

这种方法未来应该会在ai for science领域发挥潜力。

11.这个范式同样可以抽取规则，即给出输入和输出，输出变换规则，我已经在元胞自动机任务上进行了实验，非常成功。但是那个实验我通过对数据集样本生成的控制，保证了输入和输出可以唯一确定一个变换规则和演化层数。但在现实世界问题里可能往往不可能满足单样本抽取变换规则的条件，这里我的设想是，如果要通过大量输入和输出数据抽取规则，也许可以采用不同种类的投票方式，都有可能会达到好的效果。

12.如果可以对模型权重和拟合的规则的关系有一些认识，是否有可能主动地手动设计权重去模拟某一个规则？当然很有可能这是非常困难的，即使清晰的符号规则让它看起来简单了点。所以可能用训练来得到权重还是更好的方法。这个问题的价值可能主要是在理论上。

13.在训练中观察到一种现象，即有时候当损失降低到非常接近0的时候，会突然一个跳变到高位loss，推测可能是因为这类任务的非平滑性导致。

14.一个设想：完全可以把这种训练变成一种自动化的东西，人只需要和ai协同编写训练集脚本，后面的数据集生成，训练，分析都可以自动完成，这样研究会非常快。而这个每种任务对应可学与否，可学难度怎样，会变成一种新的数据，供算法科学家甚至数学家研究。

15.关于这个范式限制为合成数据，我的想法是其实我并没有对模型进行什么改变或者创新，一旦数据变为有噪声的真实世界数据，这个范式的前提就不再存在了。所以这个范式可能不能直接用于现实世界的复杂问题，它的价值可能在于提醒我深度学习神经网络本身就有拟合符号规则的能力，而且会给我提供一个研究神经网络行为的绝佳环境。也许当我对神经网络的本质更加了解后，会对当下LLM的幻觉问题，cv中模型在一些场景的失效有更深刻的理解，从而有能力打造更加可控可解释的ai。

此外，这个范式可能会对符号世界的一些问题有更加直接的应用。

16.关于最近苹果的论文，[15] The Illusion of Thinking: Understanding the Strengths and Limitations of Reasoning Models via the Lens of Problem Complexity，在这篇论文中，叙述了一些大模型无法可靠推理的问题。我在4个问题，汉诺塔，积木问题，过河问题，跳棋问题上，用我的范式进行了训练，都取得成功，尽管汉诺塔问题存在上面所说的一些微妙之处，积木问题的收敛也不够完美。详细情况请见开源代码。

另外，当时我注意到有一些反对的声音，说是因为token不足的限制所以无法正常推理。所以，我针对汉诺塔问题，设计了一个格式 $1>2$ 表示把第一个柱子最上方的盘子向第二个柱子移动，以分号分割。然后让大模型用这个格式来回答汉诺塔问题的解，结果发现基本上 $n=6$ 左右就是极限了，再往上就会出现问题，所以我认为大模型在不调用工具（如python）的情况下确实无法对这些问题进行有效推理。检验这个格式的答案是否正确的脚本，我也开源了，见代码仓库eval_hanoi.py。

17.关于这个范式含义的重新思考：这个范式似乎与计算机视觉领域在[1] AlexNet出现前的状况惊人地相似。当时，主流方法依赖于手工设计的特征（如[18] SIFT, [19] HOG），这可以看作是一种对视觉世界的“先验符号知识”。而AlexNet的成功，恰恰在于用端到端的学习（归纳）取代了这种手工偏置。而在推理领域，当前的神经符号计算，无论是改造架构还是嵌入逻辑，其本质都是将人类对推理过程的理解作为一种先验知识注入模型，而这种事先确定的结构，这种归纳偏置可能是不真实或不合适的。

第六幕：超越架构的普适性

在这一节中，我探索了深度学习的其他模型是否具有和transformer同样的能力。我总共探索了5种，mlp, rnn/lstm, cnn, unet, diffusion。

我使用mlp和rnn/lstm来拟合符号版本的元胞自动机和接雨水算法题，当然这里的输入不再是transformer类似的token，而是实数。

cnn用来拟合输入图像，输出符号的元胞自动机数据集和图像版接雨水数据集。

unet和diffusion用来拟合输入和输出都是图像的元胞自动机数据集。

数据集生成脚本：

符号版元胞自动机：generate_cellular_automata_1d.py

符号版接雨水：generate_trapping_rain_water_decoupled.py

图片版元胞自动机: generate_cellular_automata_image_and_label.py

图片版接雨水: generate_trapping_rain_water_image_to_symbol.py

6.1 元胞自动机

数据集介绍

均为1维36位元胞自动机数据集，数据集大小为300000，演化规则为110，演化层数为2，唯一的例外是，mlp模型可能过于适合元胞自动机这个任务，拟合能力过强，我加了演化层数到6。

模型结构见开源代码。

6.1.1 cnn模型

训练代码: train_convnext.py

我采用了 [5] ConvNeXt模型，这是训练过程。

1000step 训练损失: 0.49467600 | 验证损失: 0.27210647 | 验证位准确率: 84.55833333% | 验证完全匹配率: 0.10000000%

2000step 训练损失: 0.11814510 | 验证损失: 0.05764289 | 验证位准确率: 97.36018519% | 验证完全匹配率: 36.80000000%

3000step 训练损失: 0.04807972 | 验证损失: 0.03248511 | 验证位准确率: 98.63425926% | 验证完全匹配率: 56.20000000%

4000step 训练损失: 0.00763491 | 验证损失: 0.00266492 | 验证位准确率: 100.00000000% | 验证完全匹配率: 100.00000000%

5000step 训练损失: 0.00186690 | 验证损失: 0.00128128 | 验证位准确率: 100.00000000% | 验证完全匹配率: 100.00000000%

可以看到以极快的速度完成了收敛，最后在4000step的时候就已经达到了验证集完全匹配的100%正确率，这证明了至少在这个问题上，cnn有符号规则学习能力。

6.1.2 mlp模型

训练代码: train_mlp.py

对于一个巨型mlp来说，元胞自动机的演化层数为2的数据集，收敛地过于迅速，我不得不重新产生一个演化层数为6的数据集。事实证明也非常迅速地在5个epoch结束的时候达到了验证集完全匹配的100%正确率。

初步分析是因为mlp对于神经元的超距联系保存的实在是太好了，因此在这个问题上可能不会比transformer差。

6.1.3 rnn模型

训练代码: train_lstm.py

这里有一些小小的设计，一是我必须在第一个时间步把元胞自动机的输入状态完全输入，而不是一个时间步输入一位，二是，我不可能采取自回归式输出，所以我必须把某一个时间步的输出当作结果，三是，我最后刻意选取了时间步和元胞自动机的演化步数不同，因为元胞自动机的演化步数为2，所以我选择rnn模型的输出时间步为3。虽然时间步和演化步数完全一致肯定是最理想的模型设计，但是我这里想要证明的是我前面的观点，就是模型并不是以人类通常认为的逻辑或者算法步骤来执行任务，在这个实验设定下，模型被迫学习一个变换函数，它连续重复变换3次等于元胞自动机的规则110变化2次。这个设定的学习成功可以为我对这个范式的观点提供有力支持。

而实验的结果也证明，在1000step就完成了收敛，已经达到了验证集完全匹配的100%正确率。

后续我把3个时间步改为5/7/9，同样可以在很短时间内达到验证集完全匹配的100%正确率。

6.1.4 unet模型

训练代码：train_unet.py

采用mseloss，但很快就卡在0.0915的高位完全不再下降，可能是架构本身不适合这个任务，有待进一步分析。

6.1.5 diffusion模型

训练代码：train_diffusion.py

在loss变得非常低之后，生成图像仍然不对，但是可以看到diffusion学会了生成棋盘格类似的图像，只不过它每一次eval时的eval image的棋盘格的黑白格点模式完全不同，应该也是不适合做这个任务，另外，diffusion的这种迭代式的生成图像，很有可能不适合这个范式的任何任务。

6.2 接雨水

任务描述：源自LeetCode经典算法题，[42. 接雨水 - 力扣 \(LeetCode\)](#)

格式采用和上面实验部分相同的解耦输出格式。不同的是，把柱子数调整为12，则输入输出都为 $12 \times 3 = 36 = 6 \times 6$ 。方便cnn相关实验输入一个 6×6 的黑白棋盘格图像。

6.2.1 mlp模型

训练代码：train_mlp.py

最佳结果： Validation Loss: 0.0002, Bit Acc: 99.99%, Exact Match: 99.90%

6.2.2 rnn模型

训练代码：train_lstm.py

我这里选择时间步为3

最佳结果： Validation Loss: 0.0001, Bit Acc: 100.00%, Exact Match: 100.00%

6.2.3 cnn模型

训练代码：train_convnext.py

我采用了 [5] ConvNeXt模型。

最佳结果：Epoch 6 | 训练损失: 0.00003849 | 验证损失: 0.00001896 | 验证位准确率:
100.00000000% | 验证完全匹配率: 100.00000000%

6.3 总结和分析

我用元胞自动机实验证明符号规则推演能力，接雨水证明算法拟合能力。而这一系列实验证明了这个范式并不是transformer的专属，其他很多模型也有类似的能力，这就引出一个必然的结论，那就是这个范式是联结主义的深度学习神经网络的固有能力。而且，很有可能transformer也未必是最佳模型，或者说对于这个范式的不同问题，最佳模型也会不同。此处限于时间精力，我个人无力再进行探索。

第七幕：跨越鸿沟：从理想规则到现实世界的连续谱

这一节，我主要讨论两个问题。在以上所有讨论内容里，我集中精力在有精确变换规则的数据集中。所以在这一节的第一部分，我会对有精确规则的数据集的输入输出进行随机扰动，并观察训练进程和最终loss收敛情况。

第二部分，我设计了一个实验用以证明，这个范式所暗示的神经网络本质具有的规则学习能力和内插能力并不矛盾，而是可以同时存在的。

7.1 精确规则的扰动

我选择一维元胞自动机的任务，演化规则选择为110，演化层数选择为2，一维元胞自动机的位数选择为30位。具体的扰动做法为，在生成数据的过程中，对输入或输出以一定的概率进行随机翻转（即1->0或者0->1），为了简单，我只测试了单独翻转输入或单独翻转输出的情形。验证集采用不进行随机翻转的精确规则的元胞自动机数据集。

数据集代码：generate_cellular_automata_1d_perturbed.py

训练代码：train_tiny_transformer.py

7.1.1 对输出的随机翻转测验

随机翻转概率	eval loss稳定值
0.1%	0.000692
1%	0.008490
10%	0.094094

7.1.2 对输入的随机翻转测验

随机翻转概率	eval loss 稳定值
0.1%	0.001895
1%	0.021377
10%	0.230316

7.1.3 总结讨论

可以看到不管是输入还是输出扰动，都会随着扰动概率的增大而增大任务学习的难度的，这也是很自然的，另外输入的扰动，比输出的扰动影响更大，这可能是因为规则任务本身固有的非平滑性质，可以猜测如果把演化层数增加，输入的扰动还会使任务学习变得更加困难。所以，粗略地说，从无噪声精确变换规则数据集到真实世界的数据集，中间可能存在一个连续谱。

7.2 内插和规则学习的一体性

这一节用一个实验来证明神经网络的规则学习能力和内插能力不是互斥的，而是一体的。

7.1.1 实验描述

为了验证神经网络在一个单一任务中融合离散规则学习与连续值处理的能力，我们设计了一个逻辑感知混合实验。该实验基于元胞自动机图像生成任务，在输入和输出图像中，把一维36位的元胞自动机状态以行优先排列成6*6的棋盘格图像，黑色代表1，白色代表0。但对输入和输出进行了如下修改：

输入端：输入图像不再是纯黑白色块，而是将代表逻辑黑与白的格点，分别替换为在两个不同灰度区间内随机取值的色块，以此引入连续的感知信息。

输出端：要求模型生成的输出图像必须同时满足两个条件：

- 逻辑正确性：格点的黑白模式必须严格遵循元胞自动机的多步演化规则。
- 感知关联性：每个格点的最终灰度值，必须根据其对应的输入格点原始灰度值和一个简单的条件函数（白则保持，黑则反转）来精确计算。具体的，假设输入格点的灰度值为x，如果输出格点和输入格点的逻辑模式相同，则输入格点的灰度值保持为x，否则为255-x。

这个设计巧妙地迫使模型必须同时看穿输入的连续灰度值以执行离散的逻辑推理，并记住这些灰度值以完成最终的连续值映射，从而在一个不可分割的任务中，同时考验其规则学习与内插两种核心能力。

数据集代码：generate_cellular_automata_1d_to_grid_image_interp.py

训练代码：train_image2image.py

loss：MSELoss

一些任务参数：

格点维度：6*6

格点大小：40*40像素

图像分辨率：240*240

元胞自动机规则：rule 110

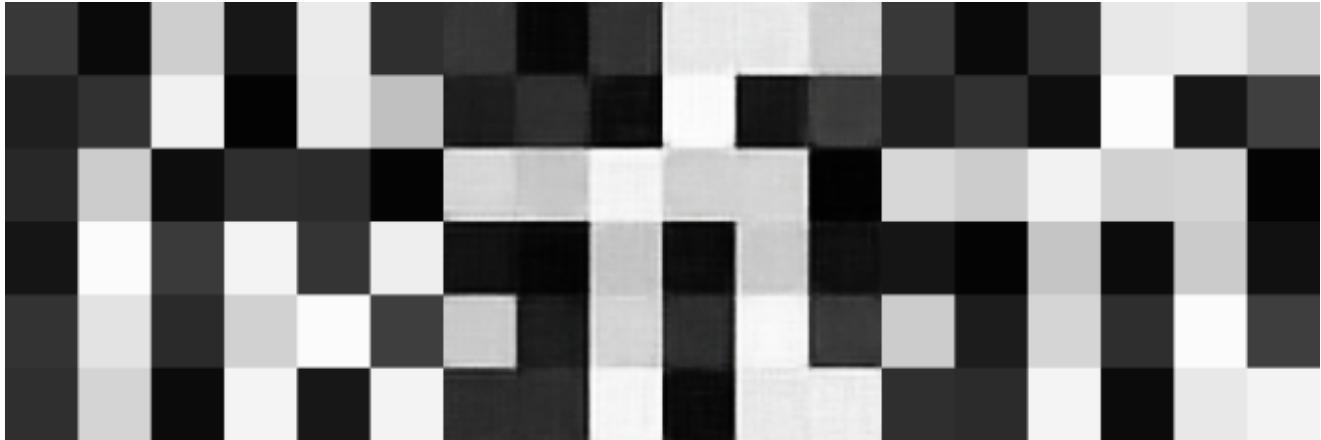
规则演化层数：2

7.1.2 实验结果

在训练3200步之后，验证集的mseloss就已经来到0.000899

这是3200步的eval image

左边是输入图像，右边是ground truth，中间是生成图像



7.1.3 总结讨论

可以看到这个实验取得了很大的成功。充分证明不管是内插还是规则学习能力，都是神经网络固有的本质能力。而且它们可能不是并行的，而是一体的。例如在这个例子中，模型并没有试图先理解元胞自动机的演化规则，然后记住每一个格点的灰度值，它实际在做的应该只是梯度下降。仍然是前面所说的，联结主义的梯度下降可以逼近离散的符号规则。

第八幕：定论——可靠AI的黎明

本研究的终点，为一个困扰人工智能领域数十年的根本性问题，提供了一个清晰而深刻的答案：一个为概率与统计而生的神经网络，是否能够真正理解并执行精确的、确定性的逻辑法则？

我的回答是肯定的。而实现这一点的关键，并非依赖于更庞大、更复杂的模型架构，而是源于一个根本性的范式转移：我摒弃了对嘈杂现实数据的模仿，转而采用由纯粹规则程序化生成的理想数据；我放弃了脆弱的自回归预测链，拥抱了并行的、一步到位的整体求解框架。

我并未止步于证明特定先进架构的优越性，而是进行了一次系统性的测试，我得到了一个颠覆性的、出乎意料的、但又无比符合第一性原理的发现：一个没有任何结构偏置的、纯粹由海量参数构成的多层感知机，竟在复杂的逻辑演化任务上，展现出了最强大的、完美的零误差泛化能力。而循环神经网络的成功，则将这一发现推向了哲学层面——它证明了神经网络甚至能够为了满足最终的数学约束，而去“创造”一个我无法直观理解的、但在功能上完全正确的迭代算法。

这一系列实验共同指向了一个不可动摇的结论：精确的、算法性的推理能力，并非任何特定架构的“专利”，它是联结主义系统本身固有的、深藏的、可以被正确范式所“唤醒”的普遍潜能。

架构的优劣，在此范式下，不再是“能与不能”的区别，而是决定了这种潜能被唤醒的“效率”与“能力边界”的区别。

这不仅仅是一项技术的突破，它是一道分水岭。它意味着，我手中第一次拥有了将抽象的、精确的、人类可理解的法则，“雕刻”进神经网络这块“大理石”的能力，而不必担心它会产生随机的“幻觉”。

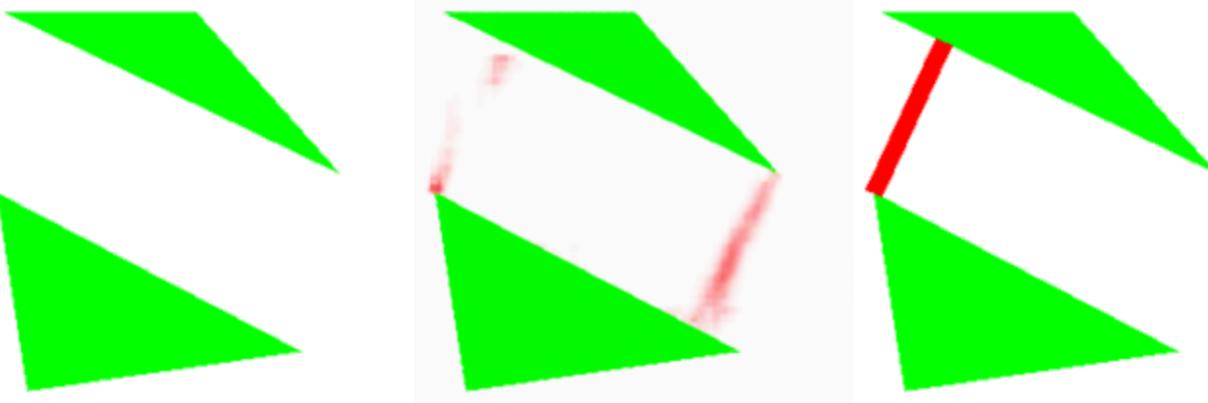
这预示着一个新时代的可能性：一个由真正可靠的、可控的、可解释的神经网络所驱动的时代。一个我不仅能与AI对话，更能信任它去执行精密计算、模拟物理世界、乃至辅助我进行最严格的科学发现的时代。

附录：

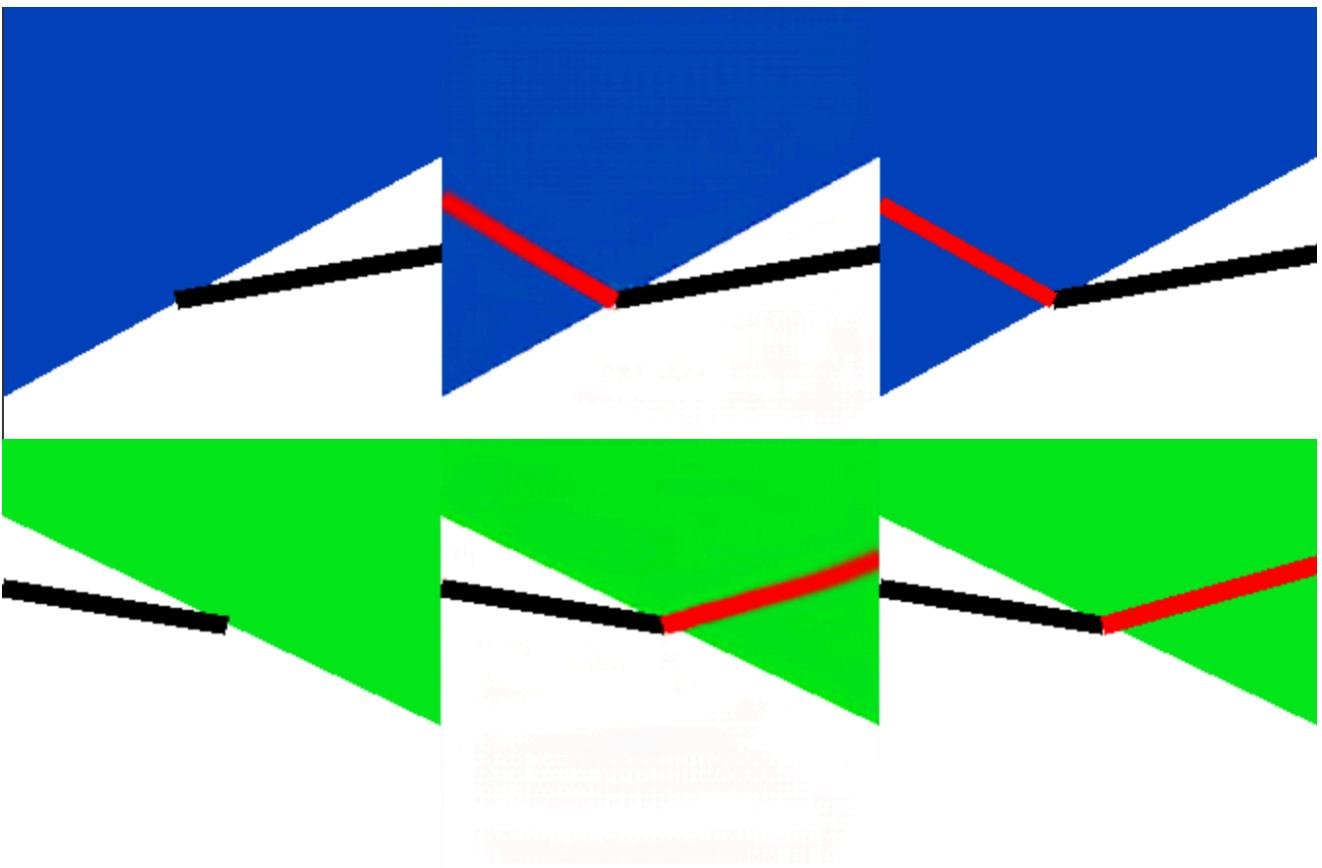
另一个任务：学习中结果的可视化

任务描述：输入图像是白色背景，两个任意不重合的绿色实心三角形，要求画一条红色线段，指示两个三角形最近距离。

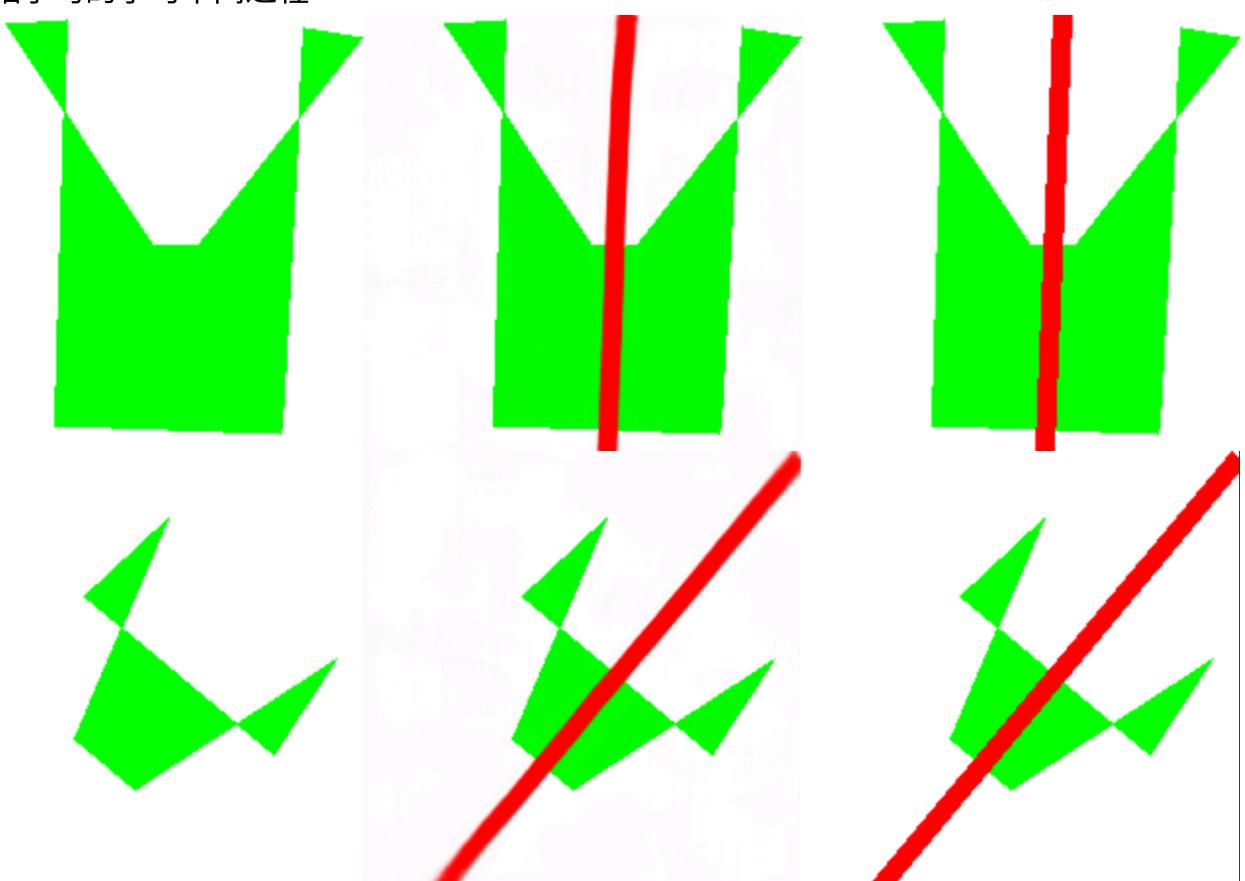
分析：似乎模型学到了最近距离的线段不是在顶点到另一个三角形对应边的垂线，就是两个顶点间的连线，但是在此处它很难确定哪个出现距离更近，因此就两个垂线都画了，但都是模糊的。左边是输入图像，中间是模型生成图像，右边是ground truth。



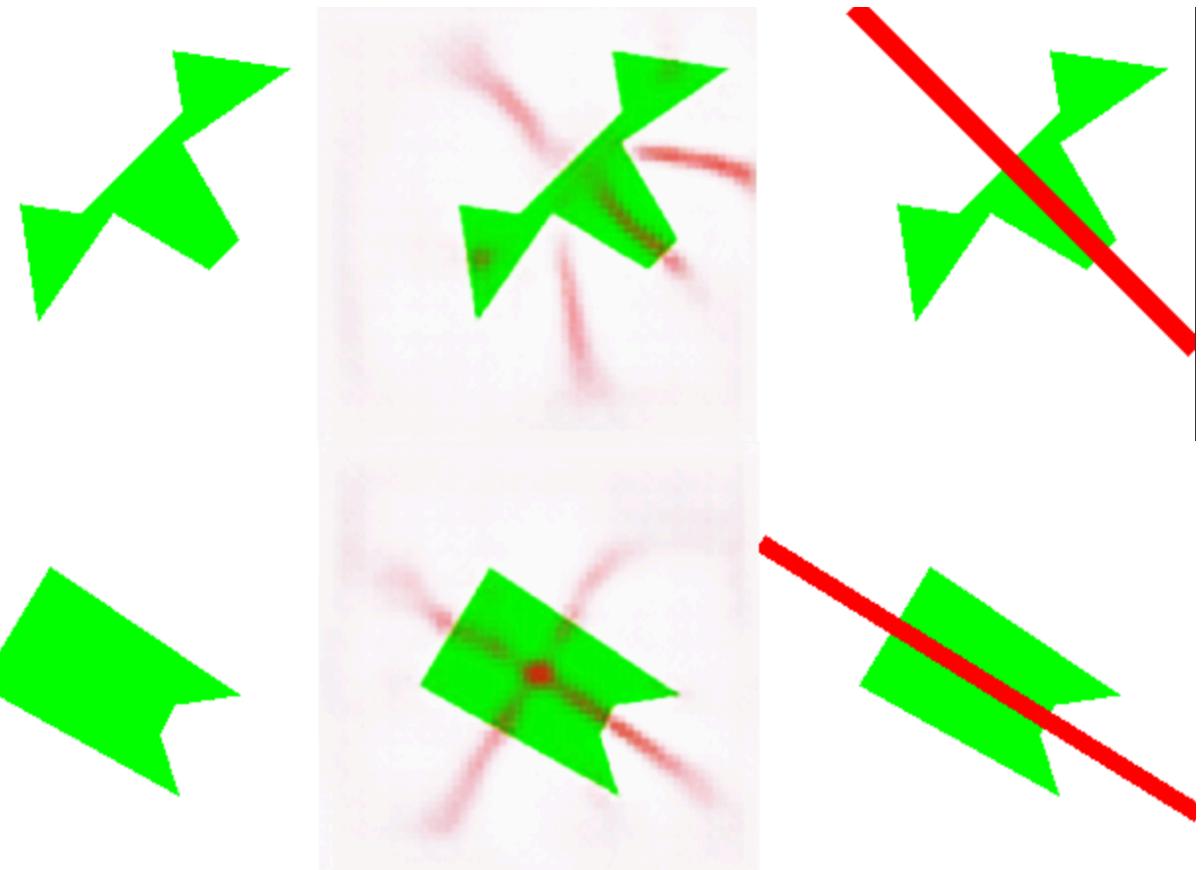
可变折射率的训练结果



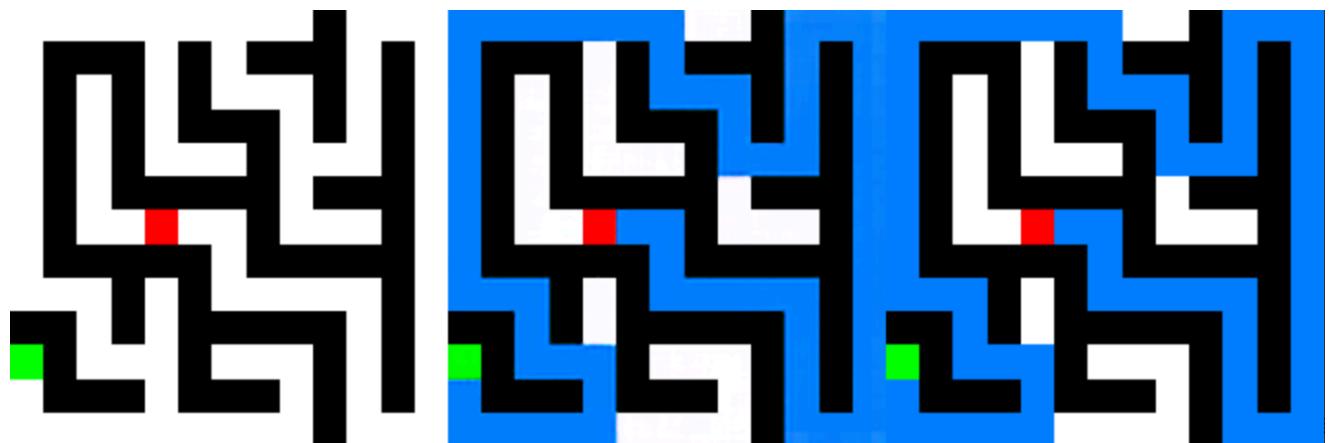
对称轴学习的学习中间过程



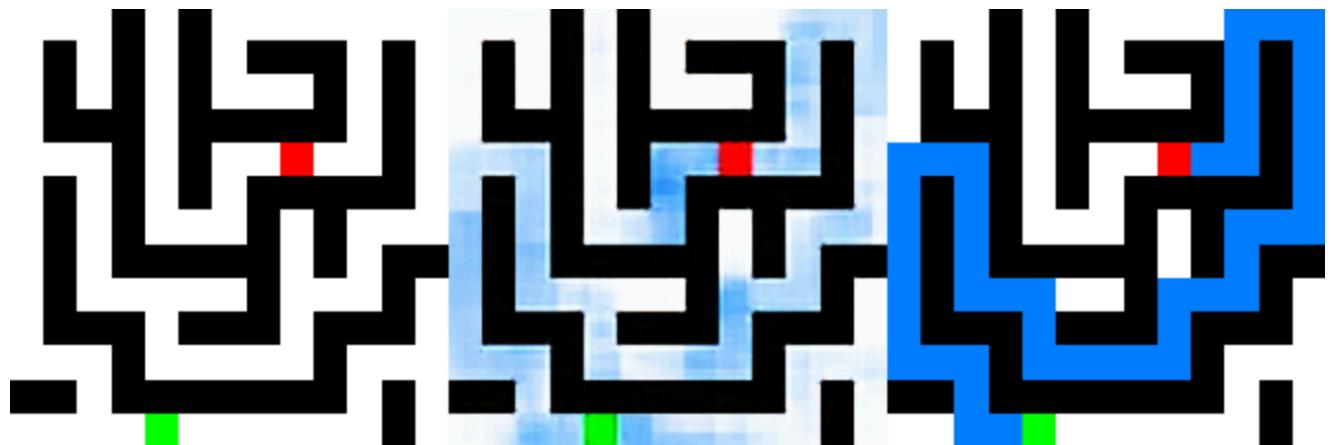
中间状态



迷宫实验



学习中间状态



最后，我发现arc-ag-3任务放出来了，改成了玩游戏的模式，本文认为可以这样子解决，一是仍然采用和之前arc-ag数据集的方法一样，我提供逻辑，用程序生成大量数据集，二是，直接把image2image的swin-unet作为state-action的policy网络，结合ppo算法，通过环境进行训练

(环境相当于潜在的无限数据)。但限于本人目前精力和时间限制，暂时还没有做这个任务。这种方式仍然需要大量数据，无法直接解决arc-agl-3。

arc-agl-3网址：[ARC-AGL-3](#)[14]

所有做过的实验：

A. 符号规则学习 (Symbolic Rule Learning)

generate_conditional_add_subtract.py: 此脚本用于探究模型处理“规则冲突”或“条件逻辑”的能力。它旨在测试当一个数据集内隐式混合了多种规则时，模型是否会学习失败；以及当提供一个明确的指示符时，模型是否能成功学习。

generate_add_binary_modulo.py: 这是一个早期的基础算术实验，用于测试模型学习模加法（或称“截断加法”）的能力，这种运算常见于计算机硬件的定宽整数运算。

generate_multiply_binary.py: 作为二进制算术能力的一个基准测试，生成N-bit整数的乘法数据集。

generate_multiply_binary_no_carry_phase1.py: 这是乘法“解耦”实验的第一阶段。旨在测试模型是否能学会乘法的第一步：无进位的按位相乘和错位相加，将一个复杂的乘法问题分解为一个更简单的计数问题。

generate_multiply_binary_from_counts_phase2.py: 这是乘法“解耦”实验的第二阶段。旨在验证一个独立的模型能否学会处理复杂的进位逻辑，即从一个“无进位计数向量”中计算出最终的二进制乘积。

generate_add_hexadecimal.py: 对比模型在不同符号系统下的学习能力。此脚本旨在验证模型学习的是加法这一抽象数学概念，还是仅仅是特定于二进制符号的模式。

generate_multiply_decimal.py: 测试模型处理非二进制符号输入（0-9字符），并执行算术运算（乘法）的能力。

generate_add_n_base_with_shuffle.py: 这是我研究中一项关键的决定性实验，旨在彻底分离模型的“表面模式匹配”能力和“抽象结构学习”能力。

generate_add_binary_with_position_shuffle.py: 这是“语义洗牌”系列实验中的“位置洗牌”部分。它旨在验证模型是否依赖于输入的固定空间结构，还是能学习到与位置无关的抽象关系。

generate_add_hidden_constant.py: 测试模型在没有任何直接线索的情况下，从大量样本中推断出隐藏规则或参数的能力。这类似于一个简化的系统辨识（System Identification）问题。

generate_multitask_alu.py: 此脚本旨在构建一个模拟算术逻辑单元（ALU）的多任务学习场景。它测试模型能否在一次前向传播中，对同一份输入并行执行多种不同的、定义明确的计算任务。

generate_modulo_operation.py: 探究模型学习模运算（Modulo Operation）的能力，这是一个在数论和计算机科学中至关重要但具有“循环”性质的运算。

generate_rsa_encryption.py: 测试模型学习高度非线性的、在计算上被认为是“困难”的确定性规则的能力。RSA加密是一个典型的例子。

generate_cellular_automata_1d.py: 用于生成一维元胞自动机（CA）的演化数据集，以测试模型学习和执行局部、确定性规则的能力。

generate_game_of_life_2d.py: 生成二维元胞自动机——Conway's Game of Life的数据集。此任务比一维CA更复杂，需要模型理解二维空间中的邻域关系。

`generate_cellular_automata_1d_multistate.py`: 作为一维元胞自动机实验的扩展，测试模型处理非二进制状态空间的能力。

`generate_cellular_automata_programmable.py`: 测试模型的“可编程性”或“元学习”能力。模型不仅要学会CA的演化过程，还要能根据每次输入中给出的不同规则来执行演化。

`generate_deduction_chain_text.py`: 生成多步逻辑推理任务，测试模型执行符号演绎 (deduction) 的能力，类似于一个简化的定理证明器。

`generate_deduction_multirule_text.py`: 测试模型在面对多个独立的、互不相干的规则时，能否根据查询 (Query) 正确地“路由”到相应的规则并进行判断。

`generate_deduction_multirule_text_v2.py`: 测试模型在面对多个独立的、互不相干的规则时，能否根据查询 (Query) 正确地“路由”到相应的规则并进行判断。

`generate_deduction_multirule_binary.py`: 这是对多规则推理任务的格式优化版本，旨在测试紧凑的二进制编码是否比稀疏的文本格式更有利于模型学习。

`generate_deduction_fixed_depth.py`: 测试模型在有明确结构、固定深度的符号演绎任务中的多步推理能力。

`generate_function_composition.py`: 测试模型学习函数组合 (Function Composition) 的能力。这要求模型像解释器一样，按顺序解析指令并对数据进行变换。

`generate_cellular_automata_inverse_rule90.py`: 测试模型解决“逆问题”(Inverse Problem) 的能力。给定一个确定性系统的输出，模型需要反向推断出满足特定约束（最稀疏且唯一）的可能输入。

`generate_count_set_bits.py`: 测试模型执行全局聚合操作的能力。与局部规则不同，计数需要模型综合整个输入序列的信息。

`generate_sum_pattern_positions.py`: 测试模型执行更复杂的、分组式的并行聚合任务的能力。模型需要先分割输入，然后对每个分割后的模式进行分类，最后对属于同一类的模式的位置信息进行累加。

`generate_sum_pattern_positions_v2.py`: 测试模型执行更复杂的、分组式的并行聚合任务的能力。模型需要先分割输入，然后对每个分割后的模式进行分类，最后对属于同一类的模式的位置信息进行累加。

`generate_sum_pairwise_hamming_distance.py`: 测试模型执行一个需要两层嵌套聚合操作的复杂任务。模型需要先在每个比特位上进行全局统计，然后再将所有比特位的结果累加起来。

`generate_circular_shift.py`: 测试模型学习位移操作的能力，特别是循环位移 (circular shift)，这是密码学和底层编程中的常见操作。

`generate_multiply_matrix_3x3.py`: 测试模型学习结构化代数运算 (矩阵乘法) 的能力，这比简单的标量运算需要更复杂的“数据路由”和“乘积累加”能力。

`generate_evaluate_boolean_expression_text.py`: 测试模型解析一个简单的领域特定语言 (DSL) 并执行求值的能力，这比前面固定结构的表达式求值更进了一步。

`generate_evaluate_arithmetic_expression.py`: 训练模型执行符号表达式的求值任务，这要求模型理解运算符优先级、变量替换和算术运算。

`generate_evaluate_arithmetic_expression_no_multiply.py`: 这是对表达式求值任务的简化版本，旨在通过移除乘法运算来降低学习难度。

`generate_evaluate_arithmetic_expression_no_multiply_small_range.py`: 这是在前一个“无乘法”版本基础上的进一步简化，通过缩小数值范围来进一步降低学习难度。

`generate_check_boolean_equivalence.py`: 测试模型对布尔代数逻辑等价性的判断能力。这是一个抽象的符号推理任务，要求模型理解表达式的结构和布尔运算法则。

`generate_polynomial_shift_coefficients.py`: 测试模型学习一个抽象的代数变换规则的能力，该任务需要模型理解多项式展开的内在结构。

`generate_convolution_2d.py`: 测试模型学习二维卷积（Conv2D）这一基本图像处理操作的能力，并探究其是否能从输入输出对中推断出隐藏的固定规则（即卷积核本身）。

`generate_simple_block_cipher.py`: 测试模型“破解”或学习一个简单但非平凡的自定义加密算法的能力。该任务代表了一类复杂的、具有高度混沌和雪崩效应的符号变换规则。

`generate_sin_function_float32.py`: 测试模型拟合连续、周期性、非线性函数 ($\sin(x)$) 的能力，使用标准的32位浮点数格式进行输入和输出。

`generate_sin_function_float64_to_int12_deprecated.py`: 这是对`sin`函数拟合任务的另一种编码尝试，旨在探索使用更高精度的浮点输入和更低精度的量化二进制输出对学习效果的影响。

`generate_sin_function_float32_to_quantized_int.py`: 测试模型拟合连续、周期性、非线性函数 ($\sin(x)$) 的能力，并探索不同输入/输出编码方案对学习效果的影响。

`generate_multiply_binary_modulo.py`: 作为基础算术实验的一部分，测试模型对截断乘法（或称模乘法）的掌握能力。

`generate_explainable_two_step_calculation.py`: 测试模型输出计算“中间步骤”或“思维链”的能力，是“功能性可解释性”的一个直接验证。

`generate_chess_positions_by_random_moves.py`: 通过模拟一个完全随机的玩家下棋的过程，快速生成大量看起来合理的、合法的中国象棋局面。

`generate_chess_positions_by_random_placement.py`: 通过在棋盘上随机放置棋子（而非模拟下棋）来生成大量非典型的、但大部分合法的中国象棋局面，用于对模型的鲁棒性进行压力测试。

`generate_chess_positions_from_engine_self_play.py`: 生成大量高质量、符合实战逻辑的中国象棋局面（FEN格式），作为训练棋类AI的基础数据源。

`generate_preprocess_legal_moves.py`: 这是一个数据预处理脚本，用于将FEN格式的局面数据集转换为模型可以直接学习的“合法走法预测”任务。

`generate_chess_resolve_check_task.py`: 生成一个专门针对中国象棋中“解将”（Resolving a Check）这一特定战术场景的数据集。这个任务要求模型在处于被将军的状态下，找出所有能够合法解除将军的走法。

B. 算法学习 (Algorithm Learning)

`generate_sort_integers.py`: 测试模型执行基本排序算法的能力，这是一个非局部的、需要对输入元素进行比较和重排的经典算法任务。

`generate_edit_distance.py`: 测试模型学习解决动态规划问题的能力。编辑距离是一个典型的DP问题，需要模型在概念上构建一个二维的求解矩阵。

`generate_edit_distance_explainable.py`: 这是“功能性可解释性”的一个核心实验。它要求模型不仅给出最终答案（编辑距离），还要输出达成答案的完整“思维链”（编辑过程）。

`generate_maze_random_walls.py`: 测试模型在随机生成的“多孔”迷宫中的基础寻路能力。

`generate_maze_dense.py`: 测试模型在复杂的、类似人类设计的“稠密”迷宫中进行路径规划的能力，这比随机墙壁迷宫更具挑战性。

`generate_blocks_world_arbitrary_goal.py`: 解决经典的“积木世界”（Blocks World）规划问题，这是AI规划领域的基准任务。此版本允许指定任意的初始状态和终止状态。

`generate_blocks_world_fixed_goal.py`: 这是对“积木世界”任务的简化，通过固定目标状态，旨在测试模型在目标明确、状态空间更结构化的情况下的学习能力。

`generate_blocks_world_fixed_goal_multilabel.py`: 进一步改进“积木世界”任务，通过允许多个最优解，测试模型处理多标签分类问题的能力，更真实地反映了规划问题中可能存在的等效最优路径。

`generate_blocks_world_fixed_goal_multilabel_fixed_format.py`: 这是“积木世界”任务的最终优化版本，通过改进输入表示法，旨在为模型提供一个更清晰、更结构化的学习目标。

`generate_checkers_jump_1d.py`: 解决一个在一维空间中移动棋子的规划问题，该问题源自苹果公司的一篇著名论文，用于测试大语言模型的推理瓶颈。

`generate_river_crossing_puzzle.py`: 解决一个经典的约束满足和状态空间搜索问题——“N对伴侣过河”。该任务源自苹果公司的一篇论文，用于揭示大型语言模型在某些类型推理任务上的局限性。

`generate_trapping_rain_water_aggregate.py`: 这是解决“接雨水”算法问题的初步尝试，旨在测试模型学习一个聚合输出（而非解耦输出）的能力。实验结果表明，要求模型直接输出总和值比输出每个位置的详细信息要困难得多。

`generate_trapping_rain_water_decoupled.py`: 解决经典的“接雨水”算法问题（LeetCode Hard）。这个任务的成功展示了模型学习需要全局信息的复杂算法的能力，并通过问题解耦的思想，证明了输出格式设计对模型学习效率的巨大影响。

`generate_trapping_rain_water_2d.py`: 作为一维“接雨水”问题的扩展，解决二维版本的“接雨水”问题。该任务要求模型理解二维空间中的“包围”和“边界”概念，是一个更复杂的全局信息处理挑战。

`generate_skyline_max_height_aggregate.py`: 这是解决“天际线”问题的初步尝试，要求模型从所有建筑的最终高度中，只预测出那个最高的高度值。此任务用于对比聚合输出和解耦输出的学习难度。

`generate_skyline_all_heights_decoupled.py`: 测试模型解决一个带有一维空间约束的全局优化问题的能力。问题原型是LeetCode "Max-Height Skyline"。通过解耦输出，要求模型预测每一栋建筑的高度，而非仅仅是最大值。

`generate_hanoi_tower_path_strategy_sep_format.py`: 这是汉诺塔问题的早期实验脚本，旨在测试模型能否学习最优路径上的策略。它采用了分隔符式的输入格式，并将动作预测为一个6分类问题。

`generate_hanoi_tower_global_strategy_fixed_format.py`: 作为对早期汉诺塔实验的改进，此脚本采用了对模型更友好的固定槽位输入格式，旨在验证输入表示对学习效率的影响。

`generate_hanoi_tower_compare_formats.py`: 这是一个对比实验脚本，它为同一个汉诺塔问题生成两种不同的输入格式（分隔符 vs. 固定槽位），用于系统性地评估不同数据表示法对模型学习递归策略的影响。

`generate_hanoi_tower_compare_formats_and_strategies.py`: 这是一个更全面的汉诺塔对比实验脚本。它不仅生成两种输入格式，还生成两种不同的数据集：一种只包含最优路径上的状态（“路径策略”），另一种包含所有可达状态（“全局策略”），用于探究模型在学习局部最优路径和全局最优策略上的能力差异。

`generate_hanoi_tower_build_full_state_graph.py`: 这是一个“汉诺塔问题”研究的集大成者，旨在通过多种不同的数据表示和采样策略，深度剖析模型对递归结构的理解能力。它是一个自给自足的数据工厂。

`generate_hanoi_tower_sample_from_state_graph.py`: 这是一个后处理和采样脚本，它利用`generate_hanoi_tower_build_full_state_graph.py`生成的完整知识库，来精确地提取特定类型的训练数据子集，例如“扭曲路径”或“最难部分”，用于进行更精细的消融实验。

`generate_sokoban_planning_astar.py`: 解决经典的“推箱子”(Sokoban) 规划问题。

`generate_sokoban_planning_full.py`: 解决经典的“推箱子”(Sokoban) 规划问题。这是一个高难度的AI任务，因为它涉及到在一个巨大的状态空间中进行搜索，并且动作会改变环境的状态。

`generate_sokoban_planning_claude_deprecated.py`: 这是一个早期的、逻辑更复杂的尝试，但未能稳定地生成高质量数据集。(已弃用)

`generate_min_swaps_for_checkerboard.py`: 解决一个高度约束的矩阵重排问题：通过任意交换行和列，将一个0/1矩阵变为“棋盘”模式（相邻元素不同）所需的最少交换次数。

`generate_min_flips_for_alternating_binary.py`: 测试模型解决一个基于位翻转的字符串优化问题，该问题可以被巧妙地映射为一个滑动窗口问题来求解。

`generate_min_swaps_for_checkerboard_v2.py`: 解决一个高度约束的矩阵重排问题：通过任意交换行和列，将一个0/1矩阵变为“棋盘”模式（相邻元素不同）所需的最少交换次数。

`generate_matrix_flip_strategy.py`: 解决一个矩阵优化的经典问题（最大化1的数量）。此版本旨在测试模型能否学习到一个“策略”而非最终结果。

`generate_matrix_flip_max_score.py`: 测试模型学习一个矩阵优化问题的能力，该问题需要通过两步贪心策略（先行翻转，后列翻转）来达到全局最优。该版本要求模型直接输出最终的聚合结果（分数）。

`generate_min_prefix_flips.py`: 测试模型学习一个依赖于历史状态的、顺序处理的贪心算法的能力。

`generate_min_k_bit_flips.py`: 测试模型学习一个依赖于历史状态的、顺序处理的贪心算法的能力，并且测试其能否将输入的一部分(k)作为“参数”来指导对另一部分($nums$)的处理。

`generate_min_k_bit_flips_fixed_k.py`: 测试模型学习一个依赖于历史状态的、顺序处理的贪心算法的能力。此版本中，环境参数($k=2$)是固定的、隐藏的，模型必须从数据中隐式学习。

`generate_special_binary_string_recursion.py`: 测试模型学习一个递归定义的字符串变换规则的能力。该问题(LeetCode Hard "Special Binary String")要求对输入进行递归分解和重组。

`generate_min_flips_for_chunked_binary.py`: 测试模型学习一个基于局部块($chunk$)的字符串变换优化问题的能力。

`generate_count_connected_components.py`: 测试模型对图结构的基本理解，特别是“连通性”这一核心概念。

`generate_check_graph_connectivity.py`: 这是对模型图论基础能力的又一个核心测试，任务是判断图中任意两点之间是否存在一条路径。

`generate_minimize_malware_spread.py`: 解决一个基于图论的病毒传播优化问题(LeetCode Hard "Minimize Malware Spread")。模型需要理解图的连通性，并评估移除不同节点对全局传播的影响。

`generate_count_islands_1d.py`: 测试模型在一维序列上进行模式识别和计数的能力。

`generate_largest_island_by_adding_one_cell.py`: 解决一个涉及图遍历和全局优化的算法问题(LeetCode 827)。模型需要评估所有可能的“填海”位置，并选出能使合并后岛屿面积最大的那一个。

`generate_largest_island_by_adding_one_cell_v2.py`: 解决一个涉及图遍历和全局优化的算法问题(LeetCode 827)。模型需要评估所有可能的“填海”位置，并选出能使合并后岛屿面积最大的那一个。

`generate_find_articulation_points.py`: 测试模型识别图的“割点”(Articulation Point)或“桥”(Bridge)的能力，这是一个图论中的重要概念。

`generate_nim_game_zeckendorf.py`: 这个实验旨在测试我的范式能否学习一个基于复杂数论（齐肯多夫表示法）的非直观博弈论问题。它脱离了简单的模式匹配，需要模型理解更深层次的数学结构。

`generate_longest_subsequence_constrained.py`: 测试模型处理一个混合了序列操作和数值约束的复杂优化问题的能力。

`generate_treasure_hunt_tsp.py`: 解决一个复杂的状态空间搜索问题，它结合了图的遍历（BFS）和组合优化（状态压缩DP），是算法竞赛中的经典难题。

`generate_freedom_trail_dp.py`: 测试模型学习解决一个需要动态规划和路径回溯的复杂优化问题的能力。

`generate_sum_of_subset_with_mask.py`: 测试模型根据一个二进制掩码从一个集合中选择元素并执行聚合操作（求和）的能力。

`generate_sudoku_6x6.py`: 测试模型在处理有强约束满足问题（Constraint Satisfaction Problem）——数独——上的能力。

`generate_valid_parentheses_path_randomDeprecated.py`: 这是解决“合法括号路径”问题的早期尝试。（早期探索/已弃用）

`generate_valid_parentheses_path_balanced.py`: 解决一个二维网格上的路径查找问题，但路径的合法性受到栈式结构（括号匹配）的约束。

`generate_sat_solver_text.py`: 测试模型解决一个标志性的NP完全问题——布尔可满足性（SAT）问题的能力。

`generate_sat_solver_compact_text.py`: 这是对 `generate_sat_solver_text.py` 的一个变种，采用了不同的输入编码格式来解决同样的3-SAT问题。

`generate_point_in_polygon.py`: 测试模型学习一个计算几何中的经典算法——射线法（Ray Casting Algorithm）——的能力。

`generate_shortest_path_in_matrix_bfs.py`: 测试模型在一个二维网格中，基于经典的广度优先搜索（BFS）算法寻找最短路径的能力。

`generate_sudoku_4x4_stepwiseDeprecated.py`: 旨在测试模型进行“步进式”（stepwise）推理的能力。（已弃用）

`generate_tiling_problemDeprecated.py`: 旨在测试模型解决一个经典的平铺覆盖优化问题的能力，这是一个NP-hard问题。（已弃用）

`generate_hanoi_tower_twisted_pathDeprecated.py`: 此脚本意图生成一个汉诺塔问题的“扭曲路径”数据集。（已弃用）

`generate_checkers_jump_1d_v2.py`: 解决一维空间中的棋子交换规划问题，该问题被用于揭示大型语言模型在某些类型推理任务上的局限性。

C. 图像输出符号 (Image to Symbol)

`generate_checkerboard_to_binary.py`: 这是一个基础的视觉到符号转换任务，用于测试模型从原始像素数据中解码结构化信息的能力。

`generate_line_angle_to_vector.py`: 测试模型从图像中提取精确几何信息（角度）的能力，这是一个比简单识别棋盘格更高级的视觉推理任务。

`generate_count_shapes_from_image.py`: 测试模型同时进行物体识别（形状）、属性识别（颜色）和计数（聚合）的多重视觉任务能力。

`generate_maze_symbolic_to_image.py`: 将符号化的迷宫路径规划数据集转换为图像格式，以测试视觉模型（如CNN、ViT）直接从像素进行路径规划的能力。

`generate_sokoban_symbolic_to_image_no_labels.py`: 这是一个数据转换脚本，用于将符号化的推箱子数据集（.jsonl格式）仅转换为图像格式，用于纯视觉任务或作为更复杂数据处理的中间步骤。

`generate_sokoban_symbolic_to_image_with_labels.py`: 这是一个数据转换脚本，用于将符号化的推箱子数据集（.jsonl格式）转换为一个完整的图像分类数据集，以供计算机视觉模型（如ViT, Swin Transformer）进行训练。

`generate_cellular_automata_image_and_label.py`: 通用数据集生成器。为元胞自动机（CA）任务同时生成图像格式（Img2Img）和符号格式（Img2Label）的数据，支持多进程加速。

`generate_trapping_rain_water_image_to_symbol.py`: 专用数据集生成器。为“接雨水”问题生成图像格式的数据，输入是柱子高度的网格图，输出是雨水量的符号标签。

D. 图像推理 (Image to Image)

`generate_triangle_to_incircle.py`: 这是展示“用梯度下降雕刻精确规则”的一个标志性实验。它测试模型能否学习到一个纯粹的、非平凡的几何构造规则（三角形内切圆）。

`generate_polygon_to_symmetry_axis.py`: 测试模型从一个完整的对称图形中反向推断出其隐含的对称轴的能力。

`generate_triangle_to_centroid.py`: 测试模型学习另一个基础几何概念——重心的能力。

`generate_triangle_to_tessellation.py`: 这是我范式能力的一个标志性展示。它测试模型能否学习一种无限的、基于晶格的生成规则。由于镶嵌图案的全局关联性和细节的精确性，它有力地排除了模型仅仅是靠“插值”或“记忆”来解决问题的可能性。

`generate_game_of_life_image_to_image.py`: 这是二维元胞自动机的image-to-image版本，测试模型能否直接在像素空间中执行基于局部规则的演化。

`generate_projectile_motion_simulation.py`: 测试模型学习一个简单的动态物理过程的能力。这要求模型从初始条件（位置和速度向量）推断出整个时空轨迹。

`generate_snell_refraction_simulation.py`: 测试模型学习基础物理定律（斯涅尔折射定律）的能力。

`generate_snell_refraction_with_contextual_index.py`: 测试模型学习基础物理定律（斯涅尔折射定律）的能力，并且要求模型能从图像的上下文信息（背景颜色）中推断出物理参数（折射率）。

`generate_cellular_automata_spatial_conditional.py`: 测试模型在单一模态（图像）内部分区和解析“指令”与“数据”的能力，是一种“伪多模态”或“空间条件化”的实验。

`generate_trapping_rain_water_visualizer.py`: 这是一个数据转换与可视化脚本。它的作用是将已经生成的、符号化的“接雨水”数据集转换为一个image-to-image格式的数据集，以便用视觉模型来解决同一个问题。

`generate_shortest_path_in_tree_deprecated.py`: 这是一个早期的实验，旨在测试模型从图像中寻找图上最短路径的能力。（早期探索/已弃用）

`generate_shortest_distance_between_triangles.py`: 测试模型在包含多个对象的情况下，进行全局几何关系（最短距离）推理的能力。

`generate_reaction_diffusion_deprecated.py`: 该脚本用于模拟一个反应-扩散系统，以生成复杂的、类似分形的“雪花”图案。（探索性/已弃用）

`generate_cellular_automata_multimodal_deprecated.py`: 生成一个真正的多模态数据集，用于训练能够同时理解图像输入和文本指令的模型。（已弃用）

E. 文字输出图像 (Text to Image)

generate_coords_to_triangle.py: 这是一个基础的符号到几何的渲染任务，测试模型将抽象的坐标信息转换为具体像素形状的能力。

generate_cellular_automata_1d_to_grid_image.py: 测试模型能否直接将一维的符号计算结果“渲染”成结构化的二维图像。

generate_triangle_coords_to_tessellation.py: 这是一个高级的、混合了符号指令和几何生成规则的推理任务。

generate_cube_rotation_matplotlib_deprecated.py: 旨在测试模型从抽象的姿态参数（旋转角度）推理并渲染出三维物体正确视图的能力。（早期探索版本）

generate_cube_rotation_pillow_v1.py: 旨在测试模型从抽象的姿态参数推理并渲染出三维物体正确视图的能力，采用了更底层的、渲染效果更精确的技术路线。（技术升级版本）

generate_cube_rotation_pillow_with_anchor.py: 测试模型从抽象的姿态参数推理并渲染出三维物体正确视图的能力，并通过引入“视觉锚点”来辅助模型学习。（论文中使用的最终版本）

generate_cube_rotation_pillow_wireframe.py: 测试模型在更稀疏的视觉输入下，能否仅通过线框和锚点信息来学习3D旋转。（变体实验版本）

F. 物理模拟 (Physics Simulation Image Paradigm)

generate_catenary_curve_simulation_deprecated.py: 这是我早期探索悬链线问题的脚本，旨在测试模型学习由物理定律确定的非线性曲线的能力。

generate_catenary_curve_from_points.py: 测试模型学习由物理定律（最小势能原理）唯一确定的非线性曲线（悬链线）的能力。

generate_orbital_path_from_initial_state.py: 测试模型学习更复杂物理定律（开普勒定律/万有引力定律）的能力。

G. ARC-AGI 探索 (ARC-AGI Exploration)

generate_arc_contextual_color_swap.py: 测试模型从图像的局部“上下文”或“示例”中学习规则，并将其应用到同一图像的全局数据的能力。这直接模仿了ARC-AGI测试的核心理念。

generate_arc_find_cross_pattern.py: 测试模型在包含大量噪音的情况下进行视觉模式识别（或可称作“目标检测”）的能力。

generate_arc_find_odd_one_out.py: 测试模型执行一个复杂的“异类发现”(Find the Odd One Out) 元推理任务。模型需要逐行进行模式比较，找出特例，并将其重新组合到输出中。

generate_arc_connect_colored_pairs.py: 测试模型在同一图像中识别多个独立“连接任务”并理解一种隐含的“图层”或“绘制优先级”规则的能力。

generate_arc_conditional_perpendicular_lines.py: 测试模型根据物体的属性（颜色）和全局参照物（边界线、图像边缘）来执行不同几何操作的能力。

generate_arc_column_projection.py: 测试模型识别复杂的上下文关系（“在...下方且在...范围内”）并执行条件性列操作的能力。

generate_arc_procedural_spiral.py: 测试模型执行一个迭代的、程序性的生成算法的能力。模型需要理解指令、跟踪状态（当前位置、方向、长度）并循环执行。

generate_arc_fractal_stamping.py: 测试模型理解和执行递归或分形生成规则的能力。模型需要将输入图案本身作为一个“笔刷”，根据输入图案中的“指令”进行重复绘制。

`generate_arc_flood_fill.py`: 测试模型执行经典的“洪水填充”(Flood Fill) 或“油漆桶”算法的能力。

`generate_arc_layered_fill.py`: 测试模型理解一个程序性极强的、依赖于拓扑距离和条件判断的复杂填充算法。

`generate_arc_fluid_simulation.py`: 测试模型在图像空间中学习和模拟一个具有特定规则的流体动态过程的能力。

`generate_arc_periodic_conditional_fill.py`: 测试模型学习一个复杂的、带有周期性和特殊case的条件格式化规则的能力。

`generate_arc_fill_square_holes.py`: 测试模型进行多步视觉推理的能力：首先识别“空洞”，然后判断其几何属性（是否为正方形），最后根据判断结果进行操作。

`generate_arc_conditional_recoloring.py`: 测试模型理解视觉图层和进行条件性对象属性修改的能力。

`generate_arc_sort_by_length_remap_position.py`: 测试模型执行一个“属性-位置解耦与重映射”的复杂排序任务。

`generate_arc_jigsaw_puzzle_simple.py`: 测试模型解决一个视觉匹配与变换问题的能力（早期版本），其中拼图块的尺寸是唯一的，可作为匹配捷径。

`generate_arc_jigsaw_puzzle_advanced.py`: 测试模型解决一个复杂的视觉匹配与变换问题的能力，该版本要求模型必须真正地根据形状（而非尺寸）来进行匹配。

`generate_arc_connect_path_by_sequence.py`: 测试模型解析外部指令序列，并据此在图像中执行多步、有状态的路径连接任务的能力。

`generate_arc_reflection_simulation_deprecated.py`: 旨在测试模型理解复杂的基于物理光学的规则，包括射线发射、碰撞检测、角度反射和颜色变换。（已废弃）

H. 逆推规则 (Inverse Rule Inference)

`generate_cellular_automata_inverse_rule.py`: 这个实验是检验模型逆向推理 (Inverse Reasoning) 能力的第一个尝试。我的问题是：如果模型能从规则正向推出结果，那么它能否从“输入-输出”对中反向推断出其背后的规则？

`generate_cellular_automata_inverse_rule_and_steps.py`: 这是在实现“唯一解”版本之前的一个早期版本，它同样旨在让模型学习预测规则和迭代次数。

`generate_cellular_automata_inverse_rule_and_steps_unique.py`: 这是对逆向推理任务的一次重大升级。我不仅要求模型推断出什么规则被应用了，还要推断出它被应用了多少次。

致谢

首先，我由衷地感谢我的家人。没有他们在过去数月高强度实验与论文撰写期间给予我的坚定支持与理解，这项工作将无法完成。

同时，本研究的顺利进行在很大程度上得益于前沿大型语言模型所提供的强大助力。在整个实验阶段，从gpt-4o到后期的gemini-2.5-pro，这些模型在数据集生成脚本的编写、实验思路的探讨、乃至论文初稿的撰写过程中，都扮演了不可或缺的协作伙伴角色。它们极大地加速了研究的迭代进程。

特别地，我需要感谢DeepMind团队的工作 [17] Amortized Planning with Large-Scale Transformers: A Case Study on Chess。该研究启发了我将模型的输出范式从自回归生成转向整体分类，这一转变是解锁模型在巨大问题空间内泛化能力的关键突破口，并直接促使我开始了对神经网络符号学习能力的系统性探索。

最后，由于时间所限，本文中的部分计算和细节可能存在疏漏或错误，我将非常感谢读者们的批评与指正。

关于匿名的说明

出于个人原因，我决定暂时以匿名方式发表这项工作。尽管我非常热切地希望与社区分享这些想法，但我尚未完全准备好面对这类项目可能引来的公众关注所带来的压力。我希望这项工作能够凭借其自身的价值得到评判，而非取决于作者是谁。

我保留在未来某个时刻公开身份的可能性。在此之前，我由衷地感谢您的理解与对我个人选择的尊重。我期待着能围绕这项工作本身，与您进行深入的交流。

References

1. Krizhevsky A, Sutskever I, Hinton G E. ImageNet classification with deep convolutional neural networks[C]//Proc. of the 25th International Conference on Neural Information Processing Systems. Lake Tahoe, NV: Curran Associates, 2012: 1097-1105.
2. Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[C]//Proc. of the 31st International Conference on Neural Information Processing Systems. Long Beach, CA: Curran Associates, 2017: 5998-6008.
3. Dosovitskiy A, Beyer L, Kolesnikov A, et al. An image is worth 16×16 words: Transformers for image recognition at scale[C]//Proc. of the 9th International Conference on Learning Representations. 2021.
4. Liu Z, Lin Y, Cao Y, et al. Swin Transformer: Hierarchical vision transformer using shifted windows[C]//Proc. of the IEEE/CVF International Conference on Computer Vision. Montreal, QC: IEEE, 2021: 9992-10002.
5. Liu Z, Mao H, Wu C Y, et al. A ConvNet for the 2020s[C]//Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. New Orleans, LA: IEEE, 2022: 11976-11986.
6. Graves A, Wayne G, Danihelka I. Neural Turing machines[J]. arXiv preprint arXiv:1410.5401, 2014.
7. Graves A, Wayne G, Reynolds M, et al. Hybrid computing using a neural network with dynamic external memory[J]. Nature, 2016, 538(7626): 471-476.
8. Rocktaschel T, Riedel S. End-to-end differentiable proving[C]//Proc. of the NIPS Workshop on Advances in Approximate Bayesian Inference. Long Beach, CA, 2017.

9. Manhaeve R, Dumancic S, Kimmig A, et al. DeepProbLog: Neural probabilistic logic programming[C]//Proc. of the 32nd International Conference on Neural Information Processing Systems. Montréal, QC: Curran Associates, 2018: 3749-3759.
10. Battaglia P W, Hamrick J B, Bapst V, et al. Relational inductive biases, deep learning, and graph networks[J]. arXiv preprint arXiv:1806.01261, 2018.
11. Andreas J, Rohrbach M, Darrell T, et al. Neural module networks[C]//Proc. of the IEEE Conference on Computer Vision and Pattern Recognition. Las Vegas, NV: IEEE, 2016: 39-48.
12. Chollet F. On the measure of intelligence[J]. arXiv preprint arXiv:1911.01547, 2019.
13. Moskvichev A, Odouard V V, Mitchell M. The ARC-AGI-2 benchmark: Evaluating advanced reasoning and conceptual generalization[J]. arXiv preprint arXiv:2505.11831, 2025.
14. ARC Prize. ARC-AGI-3: Interactive reasoning benchmark (preview)[EB/OL]. (2025-08-22)[2025-08-28]. <https://arcprize.org/arc-agi/3/>.
15. Moskvichev A, Odouard V V, Mitchell M. The illusion of thinking: Understanding the strengths and limitations of reasoning models via the lens of problem complexity[J]. arXiv preprint arXiv:2506.06941, 2025.
16. Wolfram S. A new kind of science[M]. Champaign, IL: Wolfram Media, 2002: 1-1197. ISBN 1-57955-008-8.
17. Ruoss A, Deletang G, Genewein T, et al. Amortized planning with large-scale transformers: A case study on chess[J]. arXiv preprint arXiv:2402.04494, 2024.
18. Lowe D G. Distinctive image features from scale-invariant keypoints[J]. International Journal of Computer Vision, 2004, 60(2): 91-110.
DOI:10.1023/B:VISI.0000029664.99615.94.
19. Dalal N, Triggs B. Histograms of oriented gradients for human detection[C]//Proc. of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. San Diego, CA: IEEE, 2005: 886-893. DOI:10.1109/CVPR.2005.177.
20. LeetCode. Trapping rain water[EB/OL]. [2025-08-28].
<https://leetcode.cn/problems/trapping-rain-water/description/>.
21. Yang A, Yang B, Hui B, et al. Qwen2 technical report[J]. arXiv preprint arXiv:2407.10671, 2024.
22. Karpathy A. Software 2.0[EB/OL]. (2017-11-11)[2025-08-28].
<https://karpathy.medium.com/software-2-0-a64152b37c35>.