Continuous Thought Machines

Luke Darlow[1], **Ciaran Regan**[1,2], **Sebastian Risi**[1,3], **Jeffrey Seely**[1] **and Llion Jones**[1]
[1]Sakana AI, [2]University of Tsukuba, [3]IT University of Copenhagen

Biological brains demonstrate complex neural activity, where the timing and interplay between neurons is critical to how brains process information. Most deep learning architectures simplify neural activity by abstracting away temporal dynamics. In this paper we challenge that paradigm. By incorporating neuron-level processing and synchronization, we can effectively reintroduce neural timing as a foundational element. We present the Continuous Thought Machine (CTM), a model designed to leverage neural dynamics as its core representation. The CTM has two core innovations: (1) **neuron-level temporal processing**, where each neuron uses unique weight parameters to process a history of incoming signals; and (2) **neural synchronization employed as a latent representation**. The CTM aims to strike a balance between oversimplified neuron abstractions that improve computational efficiency, and biological realism. It operates at a level of abstraction that effectively **captures essential temporal dynamics while remaining computationally tractable** for deep learning. We demonstrate the CTM's strong performance and versatility across a range of challenging tasks, including ImageNet-1K classification, solving 2D mazes, sorting, parity computation, question-answering, and RL tasks. Beyond displaying rich internal representations and offering a natural avenue for interpretation owing to its internal process, the CTM is able to perform tasks that require complex sequential reasoning. The CTM can also leverage adaptive compute, where it can stop earlier for simpler tasks, or keep computing when faced with more challenging instances. The goal of this work is to share the CTM and its associated innovations, rather than pushing for new state-of-the-art results. To that end, we believe the CTM represents a significant step toward developing more biologically plausible and powerful artificial intelligence systems.

Alongside this report, we release the CTM code repository that includes the model checkpoints. We also encourage the reader to view our project page for interactive demonstrations to best highlight the CTM's capabilities.

Corresponding author(s): Luke Darlow (luke@sakana.ai)

连续思维机器

卢克·达洛1, 西昂·雷根1,2, 西巴斯蒂安·里斯1,3, 贾弗·西利1 和 李龙·琼斯1 [1]Sakana AI, [2]
东京大学, [3]哥本哈根信息技术大学

生物大脑表现出复杂的神经活动，其中神经元之间的时间关系和相互作用对大脑处理信息至关重要。大多数深度学习架构通过抽象掉时间动态来简化神经活动。在本文中，我们挑战了这一范式。通过引入神经元级的处理和同步，我们可以有效地重新引入神经时间作为基础元素。我们提出了连续思维机器（CTM），这是一种旨在利用神经动力学作为其核心表示的模型。CTM有两个核心创新：（1）神经元级的时间处理，其中每个神经元使用独特的权重参数处理传入信号的历史；（2）作为潜在表示的神经同步。CTM旨在在简化神经元抽象以提高计算效率和生物现实性之间取得平衡。它在抽象层次上有效地捕捉到关键的时间动态，同时保持计算上可处理性以适应深度学习。我们展示了CTM在一系列具有挑战性的任务中的强大性能和灵活性，包括ImageNet-1K分类、解决2D迷宫、排序、奇偶计算、问答和强化学习任务。除了展示丰富的内部表示并提供自然的解释途径外，CTM还能够执行需要复杂序列推理的任务。CTM还可以利用自适应计算，对于简单的任务可以提前停止计算，而在面对更具挑战性的实例时继续计算。本工作的目标是分享CTM及其相关创新，而不是追求新的最先进结果。为此，我们认为CTM代表了朝着开发更具有生物合理性和强大性的人工智能系统迈出的重要一步。

Alongside 这份报告，我们发布了包含模型检查点的 CTM 代码仓库。我们还鼓励读者访问我们的项目页面以进行交互演示，从而最好地展示 CTM 的能力。

Corresponding author(s): Luke Darlow (luke@sakana.ai)

Contents

1 Introduction 3

2 Method 5

3 ImageNet-1K classification 11

4 2D Mazes: a setup that requires complex sequential reasoning 16

5 CIFAR-10: the CTM versus humans and baselines 20

6 CIFAR-100: ablation analysis 23

7 Sorting 24

8 Parity 25

9 Q&A MNIST 28

10 Reinforcement learning 31

11 Related work 34

12 Discussion and future work 35

13 Conclusion 36

A Glossary 43

B Method details 43

C ImageNet-1K 45

D 2D Mazes 45

E CIFAR-10 versus humans 50

F CIFAR-100 50

G Parity 51

H Q&A MNIST 53

I Reinforcement learning 54

J UMAP 57

K Recursive computation of the synchronization matrix 57

目录

1 介绍 3 2 方法 5 3 ImageNet-1K 分类 11 4 2D 迷宫：需要复杂序列推理的设置 16 5 CIFAR-10：CTM 与人类和基线的对比 20 6 CIFAR-100：消融分析 23 7 排序 24 8 奇偶性 25 9 Q&A MNIST 28 10 强化学习 31 11 相关工作 34 12 讨论与未来工作 35 13 结论 36 附录 A 术语表 43 附录 B 方法细节 43 附录 C ImageNet-1K 45 附录 D 2D 迷宫 45 附录 E CIFAR-10 与人类的对比 50 附录 F CIFAR-100 50 附录 G 奇偶性 51 附录 H Q&A MNIST 53 附录 I 强化学习 54 附录 J UMAP 57 附录 K 同步矩阵的递归计算 57

1. Introduction

Neural networks (NNs) were originally inspired by biological brains, yet they remain significantly distinct from their biological counterparts. Brains demonstrate complex neural dynamics that evolve over time, but modern NNs intentionally abstract away such temporal dynamics in order to facilitate large-scale deep learning. For instance, the activation functions of standard NNs can be seen as an intentional abstraction of a neuron's firing rate, replacing the temporal dynamics of biological processes with a single, static value. Such simplifications, though enabling significant advancements in large-scale machine learning (Goodfellow et al., 2016; LeCun et al., 2015; Wei et al., 2022), have resulted in a departure from the fundamental principles that govern biological neural computation.

Over hundreds of millions of years, evolution has endowed biological brains with rich neural dynamics, including spike-timing-dependent plasticity (STDP) (Caporale and Dan, 2008) and neuronal oscillations. Emulating these mechanisms, particularly the temporal coding inherent in spike timing and synchrony, presents a significant challenge. Consequently, modern neural networks do not rely on temporal dynamics to perform compute, but rather prioritize simplicity and computational efficiency. This abstraction, while boosting performance on specific tasks, contributes to a recognized gap between the flexible, general nature of human cognition and current AI capabilities, suggesting fundamental components, potentially related to temporal processing, are missing from our current models (Chollet, 2019; Lake et al., 2017; Marcus, 2018).

Why do this research? Indeed, the notably high performance of modern AI across many practical fields suggests the emulation of neural dynamics is unwarranted, or that explicitly accounting for temporal aspects of intelligence is anti-pragmatic. However, human intelligence is highly flexible, data-efficient, is fluid in that it extrapolates well to unseen circumstances, and exists in an open world where learning and adaptation are tied to the arrow of time. Consequently, human intelligence includes common sense, the ability to leverage ontological reasoning, transparency/explainability, and strong generalization. AI does not yet convincingly display these properties (Chollet, 2019; Hohenecker and Lukasiewicz, 2020; Marcus, 2018; Thompson et al., 2020).

For these reasons, we argue that time should be a central component of artificial intelligence in order for it to eventually achieve levels of competency that rival or surpass human brains (Cariani and Baker, 2022; Maass, 2001). Therefore, in this work, we address the strong limitation imposed by overlooking neural activity as a central aspect of intelligence. We introduce the *Continuous Thought Machine* (CTM), a novel neural network architecture designed to explicitly incorporate neural timing as a foundational element. Our contributions are as follows:

1. We introduce an **decoupled internal dimension**, a novel approach to modeling the temporal evolution of neural activity. We view this dimension as that over which thought can unfold in an artificial neural system, hence the choice of nomenclature. While internal recurrence is not a new concept in neural networks, our innovation lies in leveraging this recurrence to explicitly construct and manipulate neural activity patterns. By progressing through discrete internal ticks (Kirsch and Schmidhuber, 2021; Kirsch et al., 2022; Pedersen et al., 2024; Schwarzschild et al., 2021), this internal dimension allows the CTM to build up complex, time-dependent neural dynamics, directly addressing the biological principle that the timing of neural events is crucial for neural computation. This contrasts with traditional uses of recurrence, which primarily focus on processing sequential data rather than generating dynamic neural activity.
2. We provide a mid-level abstraction for neurons, which we call **neuron-level models** (NLMs), where every neuron has its own internal weights that process a history of incoming signals (i.e., pre-activations) to calculate its next activation (as opposed to a static ReLU, for example). This approach is simple to implement, scales well with existing deep learning architectures, and

1. 介绍

神经网络（NNs）最初受到生物大脑的启发，但仍然与它们的生物对应物有显著的区别。大脑展示了复杂的神经动力学，这些动力学会随着时间演变，但现代NNs故意抽象掉了这些时间动力学，以便促进大规模深度学习。例如，标准NNs的激活函数可以被视为对神经元放电率的一种故意抽象，用单一的静态值取代了生物过程的时间动力学。尽管这些简化在大规模机器学习方面取得了显著进展（Goodfellow等，2016；LeCun等，2015；Wei等，2022），但它们导致了对指导生物神经计算的基本原则的偏离。

在数亿年的进化过程中，生物学大脑获得了丰富的神经动力学，包括时间依赖性可塑性（STDP）（Caporale和Dan, 2008）和神经元振荡。模仿这些机制，特别是与尖峰时间及同步性内在的时间编码，是一项重大挑战。因此，现代神经网络并不依赖时间动力学来进行计算，而是更注重简洁性和计算效率。这种抽象虽然在特定任务上提升了性能，但也导致了人类认知的灵活和普遍性与当前AI能力之间的公认差距，表明我们当前的模型可能缺少一些基本组件，这些组件可能与时间处理有关（Chollet, 2019；Lake等, 2017；Marcus, 2018）。

为什么进行这项研究？确实，现代AI在许多实际领域表现出显著的高性能，这表明模仿神经动力学是不必要的，或者明确考虑智能的时间方面是不切实际的。然而，人类智能高度灵活，数据效率高，在未见过的情况下能够很好地外推，并且存在于一个开放的世界中，在这个世界中，学习和适应与时间之箭紧密相关。因此，人类智能包括常识、利用本体推理的能力、透明性/可解释性以及强大的泛化能力。AI尚未表现出这些特性（Chollet, 2019；Hohenecker 和 Lukasiewicz, 2020；Marcus, 2018；Thompson 等, 2020）。

由于这些原因，我们认为时间应该成为人工智能的核心组成部分，以便最终达到与人类大脑相媲美的能力水平（Cariani 和 Baker, 2022；Maass, 2001）。因此，在本文中，我们解决了忽视神经活动作为智能核心方面所造成的强大限制。我们引入了*Continuous ThoughtMachine* (CTM)，这是一种新型的神经网络架构，旨在明确将神经时间作为基础元素进行整合。我们的贡献如下：

1. 我们引入了一个解耦的内部维度，这是一种新型的方法来建模神经活动的时间演变。我们将这一维度视为思想在人工神经系统中展开的空间，因此选择了这一命名。虽然内部递归不是神经网络中的新概念，但我们的创新之处在于利用这种递归来显式地构建和操控神经活动模式。通过逐步推进离散的内部时钟（Kirsch 和 Schmidhuber, 2021；Kirsch 等人, 2022；Pedersen 等人, 2024；Schwarzschild 等人, 2021），这一内部维度使 CTM 能够构建复杂的、时间依赖的神经动力学，直接解决了神经事件的时间性对于神经计算至关重要的生物原理。这与传统递归的使用方式不同，后者主要侧重于处理序列数据，而不是生成动态的神经活动。2. 我们提供了一种中间层次的抽象，称为神经元级模型（NLMs），其中每个神经元都有自己的内部权重，用于处理传入信号的历史（即，前激活信号）以计算其下一个激活值（与静态的 ReLU 相比）。这种方法易于实现，能够很好地与现有的深度学习架构进行扩展。

results in the emergence of complex neural activation dynamics (i.e., neural activity), exhibiting a higher degree of variability than static activation functions (see Sections 3.2 and 5).
3. We use **neural synchronization** directly as the latent representation with which the CTM observes (e.g., through an attention query) and predicts (e.g., via a projection to logits). This biologically-inspired design choice puts forward neural activity as the crucial element for any manifestation of intelligence the CTM might demonstrate.

Reasoning models and recurrence. The frontier of artificial intelligence faces a critical juncture: moving beyond simple input-output mappings towards genuine reasoning capabilities. While scaling existing models has yielded remarkable advancements, the associated computational cost and data demands are unsustainable and raise questions about the long-term viability of this approach. For sequential data, longstanding recurrent architectures (Dey and Salem, 2017; Hochreiter and Schmidhuber, 1997; Medsker and Jain, 1999) have largely been superseded by transformer-based approaches (Vaswani et al., 2017). Nevertheless, recurrence is re-emerging as a natural avenue for extending model complexity. Recurrence is promising because it enables iterative processing and the accumulation of information over time. Modern text generation models use intermediate generations as a form of recurrence that enables additional compute during test-time. Recently, other works have demonstrated the benefits of the recurrent application of latent layers (Geiping et al., 2025; Jaegle et al., 2021; Yang et al., 2023).

While such methods bring us closer to the recurrent structure of biological brains, a fundamental gap nevertheless remains. **We posit that recurrence, while essential, is merely one piece of the puzzle.** The temporal dynamics unlocked by recurrence – the precise timing and interplay of neural activity – are equally crucial. The CTM differs from existing approaches in three ways: (1) the internal 'thought' dimension enables sequential thought on any conceivable data modality; (2) private neuron-level models enables the consideration of precise neural timing; and (3) neural synchronization used directly as a representation for solving tasks.

Useful side effects The CTM's internal recurrence is analogous to thought (hence the chosen nomenclature), where it can stop 'thinking' earlier for simpler tasks (e.g., an easily identifiable image; see Figure 5), or go deeper for more challenging tasks (e.g., a long maze; see Section 4.3), thus enabling a form of adaptive compute. In particular, the CTM achieves a kind of adaptive computation without the need for additional losses that are difficult to tune (Graves, 2016). Indeed, we observe the emergence of interpretable and intuitive problem-solving strategies, suggesting that leveraging neural timing can lead to more emergent benefits and potentially more effective AI systems. Another positive effect of being explicit about neural timing is that information can be encoded within this timing, the result of which is a greater capacity for contextualization – we design our 2D maze solving challenge to test this (Section 4).

The rest of this paper is structured as follows. In Section 2 we give the technical details of the CTM. In Section 3 through 10 we apply the CTM to image classification, 2D mazes, sort, parity, question-answering, and simple reinforcement learning tasks. Each experiment is designed to investigate specific characteristics and we compare to baselines wherever possible. In Section 12 we discuss our findings, suggesting avenues for future work. We draw final conclusions in Section 13. By explicitly modeling neural timing through the CTM, we aim to pave the way for more biologically plausible and performant artificial intelligence systems.

结果导致复杂的神经激活动力学（即，神经活动）的出现，表现出比静态激活函数更高的变异性（参见第3.2节和第5节）。
3. 我们直接使用神经同步作为CTM观察（例如，通过注意力查询）和预测（例如，通过投影到logits）的潜在表示。这种受生物学启发的设计选择将神经活动视为CTM可能展示的任何智能表现的关键要素。

推理模型和递归。人工智能的前沿面临一个关键节点：从简单的输入-输出映射向真正的推理能力转变。虽然扩展现有模型已经取得了显著的进步，但相关的计算成本和数据需求是不可持续的，并且引发了对这种方法长期可行性的质疑。对于序列数据，长期以来的递归架构（Dey和Salem, 2017；Hochreiter和Schmidhuber, 1997；Medsker和Jain, 1999）已经被基于变换器的方法（Vaswani等, 2017）所取代。然而，递归正在重新成为扩展模型复杂性的自然途径。递归之所以有前景，是因为它能够实现迭代处理并在时间上积累信息。现代文本生成模型使用中间生成作为递归的一种形式，在测试时提供额外的计算能力。最近，其他研究工作已经展示了递归应用潜在层的好处（Geiping等, 2025；Jaegle等, 2021；Yang等, 2023）。

虽然这些方法使我们更接近生物大脑的循环结构，但仍然存在一个根本性的差距。我们提出，循环虽然至关重要，但仅是拼图中的一块。循环解锁的时间动态——神经活动的精确时间和相互作用——同样至关重要。CTM 与现有方法有三种不同之处：(1) 内部的“思考”维度使我们能够在任何可想象的数据模态下进行序列思考；(2) 私有的神经元级模型使我们能够考虑精确的神经活动时间；(3) 直接使用神经同步作为解决问题的表示。

有用的功能副作用 CTM 的内部复发类似于思考（因此选择了这种命名方式），其中它可以为简单的任务（例如，易于识别的图像；参见图 5）提前停止“思考”，或者为更具挑战性的任务（例如，一个长迷宫；参见第 4.3 节）深入思考，从而实现一种适应性计算。特别是，CTM 在不需要额外难以调整的损失（Graves, 2016）的情况下实现了某种形式的适应性计算。事实上，我们观察到可解释且直观的问题解决策略的出现，这表明利用神经元时间可以导致更多的涌现性益处，并且可能使 AI 系统更有效。明确表示神经元时间的另一个积极影响是，信息可以编码在这一时间中，结果是增强了上下文化的能力——我们设计了 2D 迷宫解决挑战来测试这一点（第 4 节）。

本文其余部分结构如下。在第2节中，我们给出了CTM的技术细节。在第3节至第10节中，我们应用CTM于图像分类、2D 迷宫、排序、奇偶性、问答以及简单的强化学习任务。每个实验都旨在调查特定特征，并尽可能与基线进行比较。在第12节中，我们讨论了我们的发现，并提出了未来工作的方向。在第13节中，我们得出最终结论。通过显式地通过CTM建模神经元时序，我们旨在为更生物合理且性能更优的人工智能系统铺平道路。

2. Method

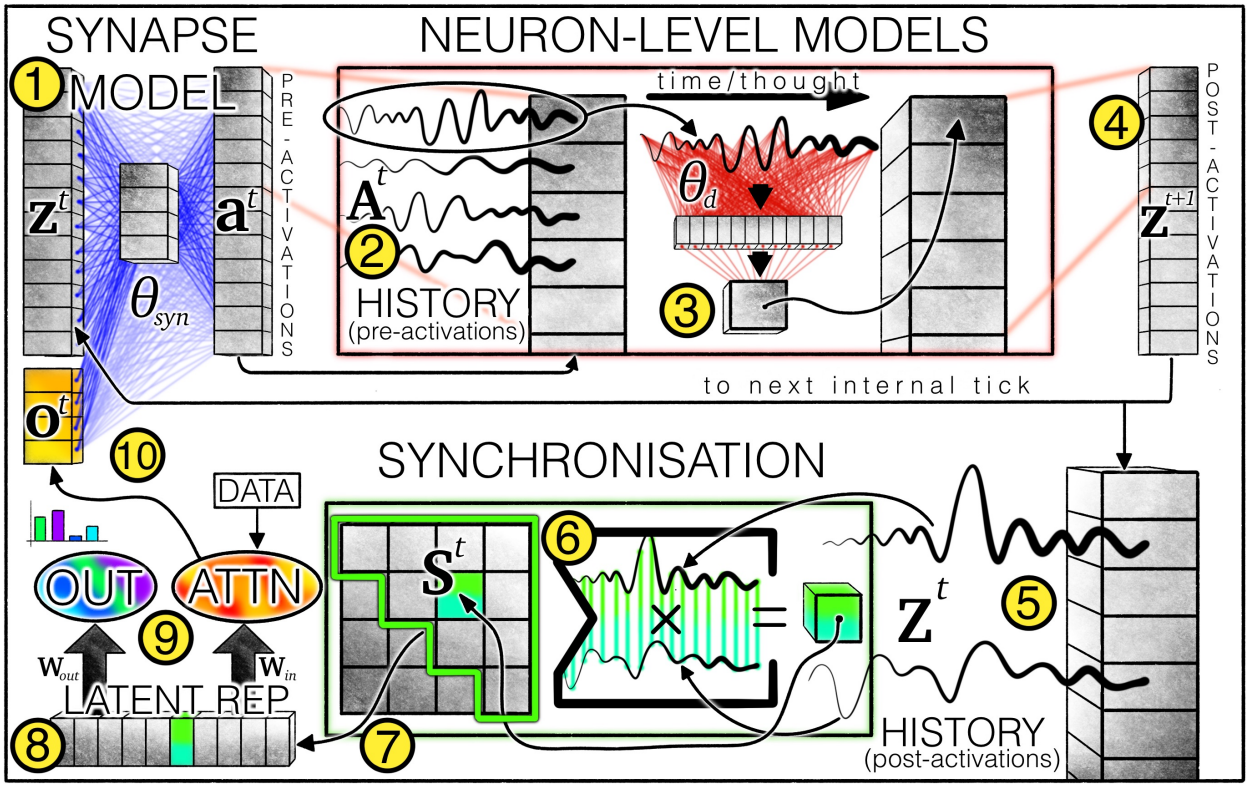


Figure 1 | CTM architecture overview. The ① synapse model (weights depicted as blue lines; Equation 1) models the cross-neuron interactions to produce pre-activations. For each neuron, a ② history of pre-activations is kept (Equation 2), the most recent of which are used by the ③ neuron-level models (weights depicted as red lines, Equation 3) to produce ④ post-activations. A ⑤ history of post-activations is also kept (Equation 4) and used to ⑥ compute a synchronization matrix (Equation 5 and Equation 10). Neuron pairs are ⑦ selected (see Section 2.4.1) from the synchronization matrix, yielding the ⑧ latent representations with which the CTM ⑨ produces outputs (Equation 6) and modulates data through cross-attention (Equation 7 and 8). Modulated data (e.g., attention outputs) are ⑩ concatenated with post-activations for the next internal tick.

The Continuous Thought Machine (CTM) is a neural network architecture that enables a novel approach to thinking about data. It departs from conventional feed-forward models by explicitly incorporating the concept of **neural dynamics** as the central component to its functionality. Figure 1 gives an overview of the CTM, with numbers ① to ⑩ designating flow, and Listing 1 gives a simplified overview listing for clarity. The yellow numbers in Figure 1 will be referred to throughout the rest of this paper.

Other recurrent architectures (e.g., RNNs) can be set up to incorporate an internal time dimension that is separate from data (Graves, 2016; Kirsch and Schmidhuber, 2021; Kirsch et al., 2022; Pedersen et al., 2024; Schwarzschild et al., 2021), but the CTM differs in two crucial ways: (1) instead of using conventional activation functions, the **CTM applies neuron-level models**, each with their own weights, to histories of pre-activations in order to produce complex neuron-level activity (see Section 3 for examples); and (2) the CTM uses **neural synchronization directly** as the latent representation when modulating data and producing outputs (see Section 2.4), effectively enabling a new depth of capacity where it creates, maintains, and leverages the exact timing and interplay of neurons. In the following subsections we will describe the CTM in detail.

2. 方法

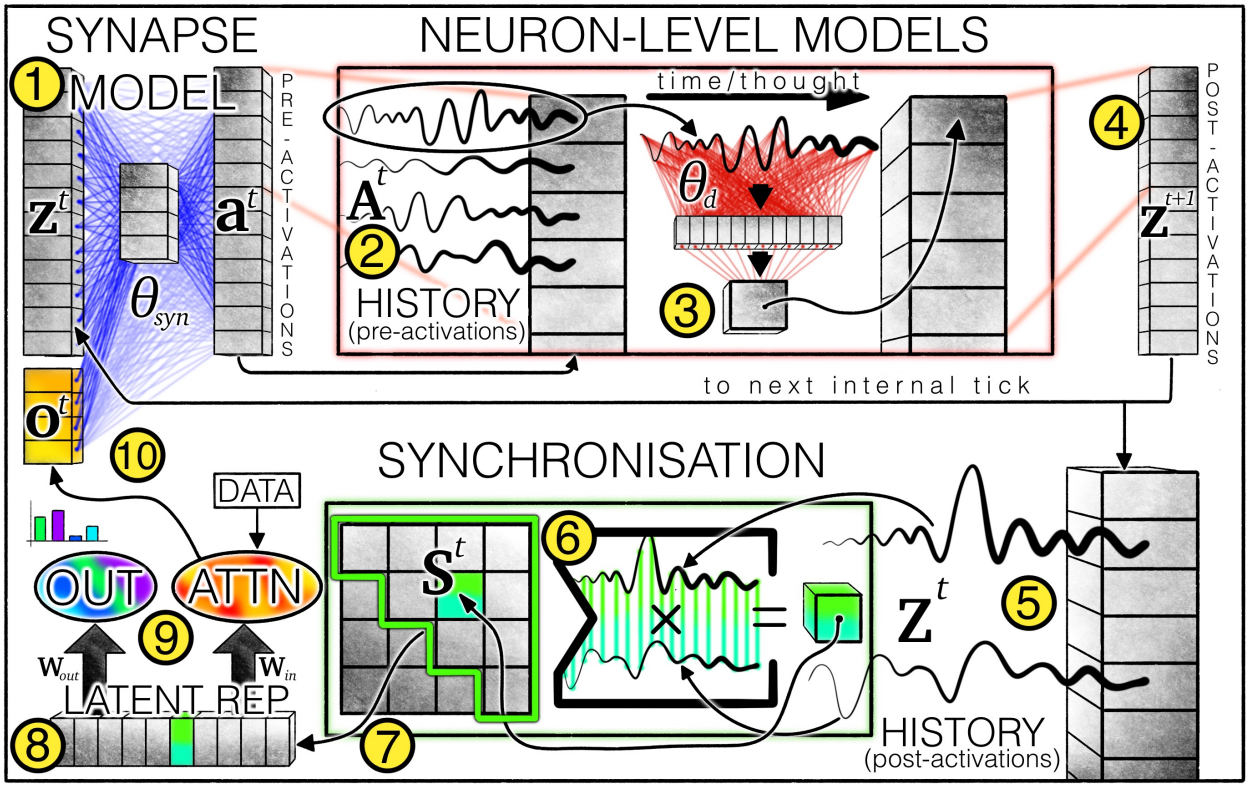


图 1 | CTM 架构概述。突触模型（权重表示为蓝色线条；方程 1）用于模拟神经元之间的交互以生成预激活。对于每个神经元，保留预激活的历史记录（方程 2），最近的预激活用于神经元级模型（权重表示为红色线条，方程 3）以生成后激活。还保留后激活的历史记录（方程 4），并用于计算同步矩阵（方程 5 和方程 10）。根据同步矩阵选择神经元对（参见第 2.4.1 节），从而生成 CTM 用于生成输出（方程 6）和通过交叉注意力调节数据（方程 7 和 8）的潜在表示。调节后的数据（例如，注意力输出）与后激活连接，用于下一个内部时钟周期。

连续思维机（CTM）是一种神经网络架构，使人们对数据进行思考的方式有了新的方法。它通过明确将神经动力学作为其功能的核心组件，与传统的前馈模型有所不同。图1给出了CTM的概览，数字表示流程，而列表1则提供了一个简化的概述列表以增加清晰度。图1中的黄色数字将在本文的其余部分中被引用。

其他循环架构（例如，RNNs）可以设置为包含一个与数据分离的内部时间维度（Graves, 2016；Kirsch 和 Schmidhuber, 2021；Kirsch 等, 2022；Pedersen 等, 2024；Schwarzschild 等, 2021），但CTM在两个关键方面有所不同：（1）CTM 不使用传统的激活函数，而是应用具有自己权重的神经元级模型来处理预激活的历史，以产生复杂的神经元级活动（参见第3节中的示例）；（2）CTM 直接使用神经同步作为潜在表示来调节数据和生成输出（参见第2.4节），这实际上使CTM能够创建、维持并利用神经元的确切时间和相互作用。在以下子节中，我们将详细描述CTM。

```
################## DEFINITIONS  (hyper parameters not shown for simplicity) ################
# Backbone can be, e.g., a ResNet for images
backbone = FeatureEncoder()
# Q and KV projectors, and standard attention module
q_projector, kv_projector = Linear(), Linear()
attn = MultiHeadAttention()
# Synapse model can be linear or MLP or U-NET
synapses = MLP()
# Neuron level models (see Listing 2)
neuron_level_models = NLMS()
# Output projector from synchronisation (see Listing 3)
output_proj = Linear()
# Initialise pre-activations and z as learnable parameters
# D is the model width
z_init = Parameter(size=(D))
# M is the neuron memory length
pre_acts_history_init = Parameter(size=(D, M))

######################################## MODEL LOGIC ########################################
# Featurise inputs with backbone and compute KV tokens
kv = kv_projector(backbone(inputs))
# In each minibatch, initialise the learnable pre_act_history
pre_acts_history = pre_acts_history_init.unsqueeze(0).repeat(B, dim=0)  # (B, D, M)
# And start the post_acts_history with the learnable z_init
post_acts_history = [z_init.unsqueeze(0).repeat(B, dim=0)]
outputs_history = []
# Get initial action synchronisation to query data
synch_a = compute_synch(post_acts_history, type="action")
# Other initialisations, including learnable start histories and first pre-attn round
for step in range(n_thought_steps):
    # Project attention query off synchronisation
    q = q_projector(synch_a)
    attn_out = attn(q, kv, kv)
    # Concatenate attention output and process via synapses
    pre_acts = synapses(concat((attn_out, z)))
    # Keep history of pre-activations. This is a FIFO structure:
    pre_acts_history = concat((pre_acts_history[:, :, :-1], pre_acts), dim=-1)
    # Compute post-activations using histories (see Listing 2)
    z = neuron_level_models(pre_acts_history)
    post_acts_history.append(z)
    # Compute synchronisations (see Listing 3)
    synch_a = compute_synch(post_acts_history, type="action")
    synch_o = compute_synch(post_acts_history, type="output")
    # Projecte prediction/output off synchronisation
    outputs_history.append(output_proj(synch_o))
# Return outputs per thought step for loss function
return outputs_history
```

Listing 1 | Simplified overview of the CTM code. Features are encoded using a backbone (e.g., ResNet layers for images), data is attended to by ⑨ projecting a query from neural synchronization, information is shared across neurons using an ① MLP synapse model to produce pre-activations, ③ private neuron-level models are applied to a ② tracked history of pre-activations (see Listing 2), synchronization is computed from a ⑤ tracked history of post-activations (see Listing 3), and outputs are ⑨ projected off of synchronization. All code is available here.

2.1. Continuous thought: the internal sequence dimension

We start by introducing the internal dimension over which cognition can occur: $t \in \{1, \ldots, T\}$, a single step of which is shown as the flow from ① to ⑩. This dimension is decoupled from the data in that it unfolds internally and is not tied to any data dimensions. Recurrence along an internal dimension is by no means a new concept (Chahine et al., 2023; Geiping et al., 2025; Jaeger, 2007), and it is garnering increased focus owing to a recent drive to build *reasoning capabilities* into modern AI. Unlike conventional sequential models – such as RNNs or Transformers – that process inputs step-by-step according to the sequence inherent in the data (e.g., words in a sentence or frames in a video), the CTM operates along a self-generated timeline of internal 'thought steps.' This internal unfolding allows the model to iteratively build and refine its representations, even when processing static or non-sequential data such as images or mazes. As a result, the CTM can engage in a thought process that is decoupled from external timing, enabling more flexible, interpretable, and biologically inspired computation. To conform with existing nomenclature used in related works (Kirsch and Schmidhuber, 2021; Kirsch et al., 2022; Pedersen et al., 2024; Schwarzschild et al., 2021), we refer to these thought steps as 'internal ticks' from here on. The CTM's internal dimension is that over which the dynamics of neural activity can unfold. We believe that such dynamics are likely a cornerstone of intelligent thought. The *for loop* in Listing 1 captures the process depicted in Figure 1.

```
################## DEFINITIONS  (hyper parameters not shown for simplicity) #################
# Backbone can be, e.g., a ResNet for images
backbone = FeatureEncoder()
# Q and KV projectors, and standard attention module
q_projector, kv_projector = Linear(), Linear()
attn = MultiHeadAttention()
# Synapse model can be linear or MLP or U-NET
synapses = MLP()
# Neuron level models (see Listing 2)
neuron_level_models = NLMS()
# Output projector from synchronisation (see Listing 3)
output_proj = Linear()
# Initialise pre-activations and z as learnable parameters
# D is the model width
z_init = Parameter(size=(D))
# M is the neuron memory length
pre_acts_history_init = Parameter(size=(D, M))

######################################## MODEL LOGIC ########################################
# Featurise inputs with backbone and compute KV tokens
kv = kv_projector(backbone(inputs))
# In each minibatch, initialise the learnable pre_act_history
pre_acts_history = pre_acts_history_init.unsqueeze(0).repeat(B, dim=0)  # (B, D, M)
# And start the post_acts_history with the learnable z_init
post_acts_history = [z_init.unsqueeze(0).repeat(B, dim=0)]
outputs_history = []
# Get initial action synchronisation to query data
synch_a = compute_synch(post_acts_history, type="action")
# Other initialisations, including learnable start histories and first pre-attn round
for step in range(n_thought_steps):
    # Project attention query off synchronisation
    q = q_projector(synch_a)
    attn_out = attn(q, kv, kv)
    # Concatenate attention output and process via synapses
    pre_acts = synapses(concat((attn_out, z)))
    # Keep history of pre-activations. This is a FIFO structure:
    pre_acts_history = concat((pre_acts_history[:, :, :-1], pre_acts), dim=-1)
    # Compute post-activations using histories (see Listing 2)
    z = neuron_level_models(pre_acts_history)
    post_acts_history.append(z)
    # Compute synchronisations (see Listing 3)
    synch_a = compute_synch(post_acts_history, type="action")
    synch_o = compute_synch(post_acts_history, type="output")
    # Projecte prediction/output off synchronisation
    outputs_history.append(output_proj(synch_o))
# Return outputs per thought step for loss function
return outputs_history
```

列表 1 | CTM 代码的简化概述。特征使用骨干（例如，对于图像使用 ResNet 层）进行编码，数据通过从神经同步投影查询来关注，信息使用 MLP 突触模型在神经元之间共享以生成预激活，私有神经元级模型应用于预激活的跟踪历史（参见列表 2），同步从后激活的跟踪历史计算得出（参见列表 3），输出从同步中投影得出。所有代码均可在此获取。

2.1. 连续思维：内部序列维度

我们首先引入认知可以发生的内部维度：$t \in \{1, \ldots, T\}$，其中的单一步骤如从到的流所示。这个维度与数据是解耦的，因为它内部展开，并不绑定到任何数据维度。沿内部维度的递归并不是一个新概念（Chahine等，2023；Geiping等，2025；Jaeger，2007），并且由于最近致力于将*reasoning capabilities*融入现代AI，它正在获得越来越多的关注。与按照数据内在顺序（例如句子中的单词或视频中的帧）逐步处理输入的常规序列模型（如RNN或Transformer）不同，CTM沿自动生成的内部“思考步骤”时间线操作。这种内部展开允许模型在处理静态或非序列数据（如图像或迷宫）时迭代地构建和细化其表示。因此，CTM可以进行与外部时间脱钩的思考过程，从而实现更灵活、可解释和生物启发式的计算。为了与相关工作中已有的术语（Kirsch和Schmidhuber，2021；Kirsch等，2022；Pedersen等，2024；Schwarzschild等，2021）保持一致，我们从这里将这些思考步骤称为“内部滴答”。CTM的内部维度是神经活动动态可以展开的维度。我们认为，这样的动态可能是智能思考的核心。Listing 1中的*for loop*捕捉了图1中所示的过程。

2.2. Recurrent weights: synapses

A ① synapse model, $f_{\theta_{\text{syn}}}$, interconnects the neurons of a shared D-dimensional latent space, $\mathbf{z}^t \in \mathbb{R}^D$, where t is the current internal tick in the recurrent unfolding of the CTM. θ_{syn} are the weights of the recurrent synapse model. The synapse model can be parameterized as a single linear projection followed by a standard activation function, or it can be a multi-layer perceptron (MLP). We found experimentally that a U-NET-esque (Ronneberger et al., 2015) MLP architecture performed better for all tasks (see Appendix B.1 for details of the synapse model). This indicates that synaptic connections benefit from additional, deeper compute. Applying the synapse model produces what we consider **pre-activations** at internal tick t:

$$\mathbf{a}^t = f_{\theta_{\text{syn}}}(\text{concat}(\mathbf{z}^t, \mathbf{o}^t)) \in \mathbb{R}^D, \tag{1}$$

where $\mathbf{o}^t$ is from input data (either directly or as the output of attention; see Equation 8), which we will explain in Section 2.4.

The M **most recent pre-activations** are then collected into ② a pre-activation 'history':

$$\mathbf{A}^t = \left[\mathbf{a}^{t-M+1} \quad \mathbf{a}^{t-M+2} \quad \cdots \quad \mathbf{a}^t\right] \in \mathbb{R}^{D \times M}. \tag{2}$$

The first M elements in the history and the first $\mathbf{z}^{t=1}$ need to be initialized. We initially experimented with initializing these two zeros (a similar strategy is undertaken for RNNs), but found that making these learnable parameters was best.

2.3. Privately-parameterized neuron-level models

```
# Initialisations
weights_1 = Parameter(shape=(M, d_hidden, d_model))
bias_1 = zeros(shape=(1, d_hidden, d_model))
weights_2 = Parameter(shape(d_hidden, d_model))
bias_2 = zeros(shape=(1, d_model))
# Forward pass
# b=batch, M=memory, d=d_model, h=d_hidden
# inputs are shape (b, d, M)
inputs = pre_acts_history[-M:]
out = einsum('bdM,Mhd->bdh', inputs, weights_1) + bias_1
out = einsum('bdh,hd->bd', out, weights_2) + bias_2
```

Listing 2 | Neuron-level models: ③. Using einsum greatly simplifies and speeds up the application of neuron-level models as their outputs can be computed in parallel. The first einsum computes a h-dimension latent for each neuron from the (truncated to M most recent) incoming history, ②. The second einsum then computes the single activation per-neuron (ignore the '1' dimension here for simplicity).

M effectively defines the length of the history of pre-activations that each neuron-level model works with. We tested a range of values for M, and found a range of 10-100 to be effective. Each neuron, $\{1, \ldots, D\}$, is then given its own ③ privately parameterized model that produces what we consider ④ post-activations:

$$\mathbf{z}_d^{t+1} = g_{\theta_d}(\mathbf{A}_d^t), \tag{3}$$

where θ_d are the unique parameters for neuron d, and $\mathbf{z}_d^{t+1}$ is a single unit in the vector that contains all post-activations. We use an MLP with a single hidden layer of width d_{hidden}. $\mathbf{A}_d^t$ is a M-dimensional vector (time series). Letting neurons have their own internal models does require additional parameters, scaling as $D \times (M \times H_{dim} + H_{dim})$ (where H_{dim} is the width of the neuron-level MLPs, ignoring bias parameters for simplicity). This additional parameter cost enables more modeling freedom, however.

The full set of neuron post-activations are then ⑩ concatenated with attention output (see Section 2.4) and fed recurrently into f_{θ_1} to produce pre-activations for next step, $t+1$, in the unfolding thought process. The recurrent thought-process employed by the CTM is captured in Listings 1 and 2. We will now discuss how the CTM interacts with data (both inputs and outputs) through synchronization.

2.2. 循环权重: 突触

一个突触模型，$f_{\theta_{\text{syn}}}$，将共享的D-维潜在空间$\mathbf{z}^t \in \mathbb{R}^D$中的神经元相互连接，其中$t$是CTM在递归展开过程中的当前内部时钟。$\theta_{\text{syn}}$是递归突触模型的权重。突触模型可以参数化为单个线性投影后跟随标准激活函数，或者是一个多层感知器（MLP）。实验结果显示，对于所有任务，一种类似U-NET（Ronneberger等，2015）的MLP架构表现更好（参见附录B.1中关于突触模型的详细信息）。这表明突触连接可以从额外的、更深的计算中受益。应用突触模型会产生我们在内部时钟t处认为的预激活：

$$\mathbf{a}^t = f_{\theta_{\text{syn}}}(\text{concat}(\mathbf{z}^t, \mathbf{o}^t)) \in \mathbb{R}^D, \tag{1}$$

w这里 $\mathbf{o}^t$ 来自输入数据（直接来自输入数据或作为注意力机制的输出；参见方程 8），我们
w我会在第2.4节解释。

The M 最近的预激活状态然后被收集到一个预激活“历史”中:

$$\mathbf{A}^t = \left[\mathbf{a}^{t-M+1} \quad \mathbf{a}^{t-M+2} \quad \cdots \quad \mathbf{a}^t\right] \in \mathbb{R}^{D \times M}. \tag{2}$$

T他首先需要初始化历史中的前 M 个元素和前 $\mathbf{z}^{t=1}$ 个元素。我们最初进行了实验
w初始化这两个零（对于RNNs采用类似策略），但发现
t这些可学习的参数是最优的。

2.3. 私有参数化神经元级模型

```
# Initialisations
weights_1 = Parameter(shape=(M, d_hidden, d_model))
bias_1 = zeros(shape=(1, d_hidden, d_model))
weights_2 = Parameter(shape(d_hidden, d_model))
bias_2 = zeros(shape=(1, d_model))
# Forward pass
# b=batch, M=memory, d=d_model, h=d_hidden
# inputs are shape (b, d, M)
inputs = pre_acts_history[-M:]
out = einsum('bdM,Mhd->bdh', inputs, weights_1) + bias_1
out = einsum('bdh,hd->bd', out, weights_2) + bias_2
```

列表 2 | 神经元级模型：. 使用 einsum 显著简化并加快了神经元级模型的应用，因为它们的输出可以并行计算。第一个 einsum 从（截断为 M 最近的）输入历史中为每个神经元计算一个 h 维潜在表示，. 第二个 einsum 然后为每个神经元计算单一激活（这里忽略 ‘1’ 维度以简化处理）。

M 有效地定义了每个神经元级模型工作的前激活历史长度。我们测试了 M 的一系列值，并发现 10-100 的范围是有效的。每个神经元 $\{1, \ldots, D\}$ 都被赋予了自己的私有参数化模型，生成我们认为的后激活：

$$\mathbf{z}_d^{t+1} = g_{\theta_d}(\mathbf{A}_d^t), \tag{3}$$

其中 θ_d 是神经元 d 的唯一参数，而 $\mathbf{z}_d^{t+1}$ 是包含所有后激活值的向量中的一个单元。我们使用单隐藏层且宽度为 d_{hidden} 的多层感知机（MLP）。$\mathbf{A}_d^t$ 是一个 M 维的向量（时间序列）。让神经元拥有自己的内部模型确实需要额外的参数，这些参数的数量与 $D \times (M \times H_{dim} + H_{dim})$ (成正比，其中 H_{dim} 是神经元级 MLP 的宽度，为了简化，忽略偏差参数。然而，这种额外的参数成本提供了更多的建模自由度。

The 完整的神经元后激活集合然后与注意力输出连接（参见第2.4节），并递归地输入到 f_{θ_1} 中以生成下一步骤的预激活，$t+1$，在展开的思考过程中。CTM 所采用的递归思考过程在代码清单 1 和 2 中被捕获。我们现在将讨论 CTM 如何通过同步与数据（包括输入和输出）交互。

2.4. Neural synchronization: modulating data and outputs

```
# AT INITIALISATION:
# Pre choose D_chosen neuron pairs from D total neurons
# D_chosen can be D_out or D_action
# Other neuron selection strategies exist, but here we show random selection
idxs_left = randint(low=0, high=D, size=D_chosen)
idxs_right = randint(low=0, high=D, size=D_chosen)  # can overlap
# Define learnable exponential decay scaling factors per neuron pair
r = Parameter(zeros(1, D_chosen, 1))
# INITIALISATION OVER.
# IN FORWARD PASS:
S = stack(post_acts_history, 1)  # S is of shape [B, T=history length]
# decay BACK in time
t_back = range(T-1, -1, -1).reshape(1, T, 1)
# Compute per NEURON PAIR exponential decays, and expand over D_chosen
exp_decay = exp(-t_back * r).expand(1, T, D_chosen)
# Compute weighted inner dot products using differet subsets of neurons
S_multiplied = S[:,:,idxs_left] * exp_decay * S[:,:,idxs_right]  # [B, T, D_chosen]
# Sum over the free T dimension and normalise by sqrt of AUC of decays
synch_representation = (S_multiplied).sum(1)/sqrt(exp_decay.sum(1))  # [B, D_chosen]
```

Listing 3 | Neural synchronization to create the latent representations used in Listing 1. Careful reshaping and broadcasting enables the use of per neuron-pair learnable exponential decays for synchronization, which lets the CTM learn complex timing dependencies. The decay parameters are initialized as zeros (i.e., no decay). This process is repeated for output and action (see Section 2.4.1). In practice we use a recursive approach that greatly reduces compute overhead; see Appendix K.

How should the CTM interact with the outside world? Specifically, how should the CTM consume inputs and produce outputs? We introduced a timing dimension over which something akin to thought can unfold. We also want the CTM's relationship with data (its interaction, so to speak) to depend not on a *snapshot* of the state of neurons (at some t), but rather on the **ongoing temporal dynamics of neuron activities**.[1] By way of solution, we turn again to natural brains for inspiration and find the concept of neural synchronization (Uhlhaas et al., 2009) both fitting and powerful. For synchronization we start by collecting the post-activations into ⑤ a post-activation 'history':

$$\mathbf{Z}^t = \begin{bmatrix} \mathbf{z}^1 & \mathbf{z}^2 & \cdots & \mathbf{z}^t \end{bmatrix} \in \mathbb{R}^{D \times t}. \tag{4}$$

The length of $\mathbf{Z}^t$ is equal to the current internal tick, meaning that **this dimension is not fixed** and can be arbitrarily large. We define neural synchronization as the matrix yielded by ⑥ the inner dot product between post-activation histories:

$$\mathbf{S}^t = \mathbf{Z}^t \cdot (\mathbf{Z}^t)^\intercal \in \mathbb{R}^{D \times D}. \tag{5}$$

2.4.1. Neuron pairing: a sub-sampling approach

Since this matrix scales in $O(D^2)$ it makes practical sense to subsample (i, j) row-column pairs, which capture the synchronization between neurons i and j. To do so we randomly select D_{out} and D_{action} (i, j) pairs from $\mathbf{S}$, thus collecting two **synchronization representations**, $\mathbf{S}^t_{\text{out}} \in \mathbb{R}^{D_{\text{out}}}$ and $\mathbf{S}^t_{\text{action}} \in \mathbb{R}^{D_{\text{action}}}$. $\mathbf{S}^t_{\text{out}}$ can then be projected to an output space as:

$$\mathbf{y}^t = \mathbf{W}_{\text{out}} \cdot \mathbf{S}^t_{\text{out}}, \tag{6}$$

and $\mathbf{S}^t_{\text{action}}$ can be used to take actions in the world (e.g., via attention as is in our setup):

$$\mathbf{q}^t = \mathbf{W}_{\text{in}} \cdot \mathbf{S}^t_{\text{action}}, \tag{7}$$

where $\mathbf{W}_{\text{out}}$ and $\mathbf{W}_{\text{in}}$ are learned weight matrices that project synchronization into vectors for observation (e.g., attention queries, $\mathbf{q}^t$) or outputs (e.g., logits, $\mathbf{y}^t$). Even though there are $(D \times (D+1))/2$ unique pairings in $\mathbf{S}^t$, D_{out} and D_{action} can be orders of magnitude smaller than this. That said, the full synchronization matrix is a large representation that has high future potential.

[1] We did begin with snapshot representations but struggled to get stable behavior owing to the emergent oscillatory behavior of neurons.

2.4. 神经同步：调节数据和输出

```
# AT INITIALISATION:
# Pre choose D_chosen neuron pairs from D total neurons
# D_chosen can be D_out or D_action
# Other neuron selection strategies exist, but here we show random selection
idxs_left = randint(low=0, high=D, size=D_chosen)
idxs_right = randint(low=0, high=D, size=D_chosen)  # can overlap
# Define learnable exponential decay scaling factors per neuron pair
r = Parameter(zeros(1, D_chosen, 1))
# INITIALISATION OVER.
# IN FORWARD PASS:
S = stack(post_acts_history, 1)  # S is of shape [B, T=history length]
# decay BACK in time
t_back = range(T-1, -1, -1).reshape(1, T, 1)
# Compute per NEURON PAIR exponential decays, and expand over D_chosen
exp_decay = exp(-t_back * r).expand(1, T, D_chosen)
# Compute weighted inner dot products using differet subsets of neurons
S_multiplied = S[:,:,idxs_left] * exp_decay * S[:,:,idxs_right]  # [B, T, D_chosen]
# Sum over the free T dimension and normalise by sqrt of AUC of decays
synch_representation = (S_multiplied).sum(1)/sqrt(exp_decay.sum(1))  # [B, D_chosen]
```

列表 3 | 神经同步以生成在列表 1 中使用的潜在表示。仔细的重塑和广播使得可以为同步使用每个神经元对可学习的指数衰减，这使得 CTM 能够学习复杂的时序依赖关系。衰减参数初始化为零（即，没有衰减）。这个过程还用于输出和动作（参见第 2.4.1 节）。实际上，我们使用一种递归方法，这大大减少了计算开销；参见附录 K。

CTM应该如何与外部世界交互？具体来说，CTM应该如何消费输入并产生输出？我们引入了一个时间维度，在这个维度上，类似于思考的过程可以展开。我们还希望CTM与数据的关系（即它的交互）不仅依赖于神经元状态（在某个t时刻）的*snapshot*，而是依赖于神经元活动的持续时间动态。为了解决这个问题，我们再次从自然大脑中寻找灵感，发现神经同步（Uhlhaas等，2009）这一概念既合适又强大。对于同步，我们首先将后激活值收集到一个后激活“历史”中：

⑤

$$\mathbf{Z}^t = \begin{bmatrix} \mathbf{z}^1 & \mathbf{z}^2 & \cdots & \mathbf{z}^t \end{bmatrix} \in \mathbb{R}^{D \times t}. \tag{4}$$

T他长度的 Z^t 等于当前内部tick，这意味着这个维度不是固定的
c它可以任意大。我们定义神经同步为由内积生成的矩阵。 ⑥
p产品之间的激活历史乘积：

$$\mathbf{S}^t = \mathbf{Z}^t \cdot (\mathbf{Z}^t)^\intercal \in \mathbb{R}^{D \times D}. \tag{5}$$

2.4.1. Neuron pairing: a sub-sampling approach

由于这个矩阵在$O(D^2)$下进行缩放，因此从(i, j)行-列对中进行下采样是有实际意义的，这些对捕捉了神经元i和j之间的同步关系。为此，我们从S中随机选择D_{out}和D_{action} (i, j)对，从而收集了两个同步表示，$\mathrm{S}^t_{\text{out}} \in \ ^{D_{\text{out}}}$和$\mathrm{S}^t_{\text{action}} \in \ ^{D_{\text{action}}}$。然后，$\mathrm{S}^t_{\text{out}}$可以被投影到输出空间：

$$\mathbf{y}^t = \mathbf{W}_{\text{out}} \cdot \mathbf{S}^t_{\text{out}}, \tag{6}$$

并且 $\mathrm{S}^t_{\text{action}}$ 可以用于在世界中采取行动（例如，通过注意力机制，如我们在设置中所做）：

$$\mathbf{q}^t = \mathbf{W}_{\text{in}} \cdot \mathbf{S}^t_{\text{action}}, \tag{7}$$

其中，W_{out} 和 W_{in} 是学习得到的权重矩阵，将同步投影到用于观察（例如，注意力查询，q^t）或输出（例如，logits，y^t）的向量中。尽管在 S^t 中有 $(D \times (D + 1))/2$ 种独特的配对方式，但 D_{out} 和 D_{action} 可以比这小多个数量级。不过，完整的同步矩阵是一个大型表示，具有很高的未来潜力。

[1]We did begin with snapshot representations but struggled to get stable behavior owing to the emergent oscillatory behavior of neurons.

In most of our experiments we used standard cross attention (Vaswani et al., 2017):

$$\mathbf{o}^t = \text{Attention}(Q = \mathbf{q}^t, KV = \text{FeatureExtractor}(\text{data})), \tag{8}$$

where a 'FeatureExtractor' model, e.g., a ResNet (He et al., 2016), is first used to build useful local features for the keys and values. $\mathbf{o}^t \in \mathbb{R}^{d_{\text{input}}}$ is the output of attention using n_{heads} heads and is (10) concatenated with $\mathbf{z}^{t+1}$ for the next cycle of recurrence. For clarity, Listing 3 demonstrates what this process looks like in code, including learnable temporal dependency scaling (see below).

Scaling temporal dependency Since $\mathbf{S}^t$ aggregates information from all previous internal ticks up to step t, later steps could potentially have a diminishing influence on synchronization. To provide the CTM with the flexibility to modulate the influence of past activity, we introduce learnable exponentially decaying rescaling factors. Given learnable scaling factors $r_{ij} \geq 0$ for each neuron pair ij, the rescaling factors over t are computed as:

$$\mathbf{R}_{ij}^t = \left[\exp(-r_{ij}(t-1)) \quad \exp(-r_{ij}(t-2)) \quad \cdots \quad \exp(0)\right]^\intercal \in \mathbb{R}^t. \tag{9}$$

$\mathbf{R}_{ij}^t$ is then used to rescale the components of the synchronization dot product:

$$\mathbf{S}_{ij}^t = \frac{(\mathbf{Z}_i^t)^\intercal \cdot \text{diag}(\mathbf{R}_{ij}^t) \cdot \left(\mathbf{Z}_j^t\right)}{\sqrt{\sum_{\tau=1}^t \left[\mathbf{R}_{ij}^t\right]_\tau}}, \tag{10}$$

Intuitively, higher r_{ij} values result in shorter-term dependency by biasing the dot product towards more recent internal ticks, while $r_{ij} = 0$ recovers the standard dot product (we also initialise to zeros). The square-root normalization prevents any single ij combination from dominating downstream processing (Vaswani et al., 2017). In practice we only need to compute $\mathbf{R}_{ij}^t$ for subsampled synchronization pairs, which we do separately for output and action representations. Since CTM can learn to alter the decay rates across different neuron pairs, the inclusion of r into the learnable parameters effectively enhances the complexity of the CTM's reliance on neural dynamics. For full disclosure, we found that the CTM never leveraged this for ImageNet classification (see Section 3), where only 3 of 8196 (i, j) neuron pairs in $\mathbf{S}_{\text{out}}^t$ had any meaningful decay. For navigating 2D mazes (see Section 4) approximately 3% of (i, j) pairs had meaningful decay, indicating that a task with clear sequential reasoning warrants a more local perspective (i.e., a faster decay to synchronization).

Recovering latent snapshot dependency Interestingly, the CTM could learn to recover a dependency on the current state $\mathbf{z}^t$, if we set $i = j$ and r_{ij} is very small, since this would be approximately equivalent to computing the element-wise square of $\mathbf{z}^t$ for each i sampled neuron. In practice this turns out to be unnecessary, but we did explore several subsampling strategies and discuss these in Appendix B.2.

2.5. Loss function: optimizing across internal ticks

The CTM produces outputs at each internal tick, t. A key question arises: how do we optimize the model across this internal temporal dimension? Let $\mathbf{y}^t \in \mathbb{R}^C$ be the prediction vector (e.g., probabilities of classes) at internal tick t, where C is the number of classes. Let y_{true} be the ground truth target. We can compute a loss at each internal tick using a standard loss function, such as cross-entropy:[2]

$$\mathcal{L}^t = \text{CrossEntropy}(\mathbf{y}^t, y_{true}), \tag{11}$$

[2] In practice any appropriate loss function could be used.

在我们的大多数实验中我们使用了标准的交叉注意力（Vaswani等，2017）：

$$\mathbf{o}^t = \text{Attention}(Q = \mathbf{q}^t, KV = \text{FeatureExtractor}(\text{data})), \tag{8}$$

其中，使用一个“FeatureExtractor”模型，例如ResNet（He et al., 2016），首先用于构建用于键和值的有用局部特征。$o^t \in$ $^{d_{\text{input}}}$是使用n_{heads}个头的注意力输出，并与z^{t+1}连接以进行下一轮递归。为了清晰起见，代码示例3展示了这一过程在代码中的实现，包括可学习的时间依赖缩放（详见下方）。

扩展时间依赖性 由于S^t聚合了从步骤t之前的所有内部时间步的信息，后续步骤对同步的影响可能会逐渐减弱。为了给CTM提供调节过去活动影响的灵活性，我们引入了可学习的指数衰减重缩放因子。对于每个神经元对ij的可学习缩放因子$r_{ij} \geq$，重缩放因子在t上的计算公式为：

$$\mathbf{R}_{ij}^t = \left[\exp(-r_{ij}(t-1)) \quad \exp(-r_{ij}(t-2)) \quad \cdots \quad \exp(0)\right]^\intercal \in \mathbb{R}^t. \tag{9}$$

R_{ij}^t 然后用于重新缩放同步点积的组件：

$$\mathbf{S}_{ij}^t = \frac{(\mathbf{Z}_i^t)^\intercal \cdot \text{diag}(\mathbf{R}_{ij}^t) \cdot \left(\mathbf{Z}_j^t\right)}{\sqrt{\sum_{\tau=1}^t \left[\mathbf{R}_{ij}^t\right]_\tau}}, \tag{10}$$

直观上，更高的 r_{ij} 值会导致更短的依赖期，因为它们倾向于使点积偏向于更近的内部时钟，而 $r_{ij} = 0$ 恢复标准点积（我们也将其初始化为零）。平方根规范化防止任何单一 ij 组合在下游处理中占据主导地位（Vaswani 等人，2017 年）。在实践中，我们只需要为下采样的同步对计算 R_{ij}^t，我们分别对输出和动作表示进行计算。由于 CTM 可以学习改变不同神经元对的衰减率，将 r 包含在可学习参数中实际上增强了 CTM 对神经动力学的依赖复杂性。为完全披露，我们发现 CTM 在 ImageNet 分类中从未利用这一点（参见第 3 节），其中 S_{out}^t 中仅有 8196 个 (i, j) 神经元对中有 3 个具有任何有意义的衰减。对于在 2D 迷宫中导航（参见第 4 节），大约 3% 的 (i, j) 对具有有意义的衰减，这表明一个具有明确顺序推理的任务需要更局部的观点（即，更快的同步衰减）。

恢复潜在快照依赖性 有趣的是，如果我们将 $i = j$ 和 r_{ij} 设得很小，CTM 就能够学习恢复对当前状态 z^t 的依赖性，因为这大约相当于对每个采样神经元 i 计算 z^t 的元素平方。实际上这证明是不必要的，但我们确实探索了几种下采样策略，并在附录 B.2 中讨论了这些策略。

2.5. 损失函数：在内部时钟上进行优化

CTM 在每个内部时钟周期生成输出，t。一个关键问题出现了：我们如何在这一内部时间维度上优化模型？设在内部时钟周期 t 时的预测向量 \(y_^t ∈ \in \mathbb{R}_^C \) 为各类别的概率（例如），其中 \(v31 \) 是类别的数量。设 \(v32 \) 为真实目标。我们可以在每个内部时钟周期使用标准损失函数（例如交叉熵）计算损失：[2]

$$\mathcal{L}^t = \text{CrossEntropy}(\mathbf{y}^t, y_{true}), \tag{11}$$

[2] In practice any appropriate loss function could be used.

and a corresponding certainty measure, C^t. We compute certainty simply as 1 - normalized entropy. We compute $\mathcal{L}^t$ and C^t for all $t \in \{1, \ldots, T\}$, yielding losses and certainties per internal tick, $\mathcal{L} \in \mathbb{R}^T$ and $C \in \mathbb{R}^T$.

A natural question arises: how should we reduce $\mathcal{L}$ into a scalar loss for learning? Our loss function is designed to optimize CTM performance across the internal thought dimension. Instead of relying on a single step (e.g., the last step), which can incentivize the model to only output at that specific step, we dynamically aggregate information from two internal ticks: the point of minimum loss and the point of maximum certainty. This approach offers several advantages: (1) it encourages the CTM to develop meaningful representations and computations across multiple internal ticks; (2) it naturally facilitates a curriculum learning effect, where the model can initially utilize later internal ticks for more complex processing and gradually transition to earlier steps for simpler processing; and (3) it enables the CTM to adapt its computation based on the inherent difficulty of individual data points within a dataset. To this end, we aggregate $\mathcal{L}$ dynamically (per data point) over two internal ticks:

1. the point of minimum loss: $t_1 = \text{argmin}(\mathcal{L})$; and
2. the point of maximum certainty: $t_2 = \text{argmax}(\mathcal{C})$.

The final loss is then computed as:

$$L = \frac{\mathcal{L}^{t_1} + \mathcal{L}^{t_2}}{2}, \tag{12}$$

and stochastic gradient descent is used to optimize the model parameters, θ_{syn} and $\theta_{d=1\ldots D}$ (see Equations 1 and 3).

This approach effectively enables the CTM to improve its 'best' prediction while making sure that high certainty is attributed to correct predictions. Listing 4 shows how the loss is calculated, demonstrating how the certainty is computed as 1 - normalized entropy. This approach also enables the CTM to dynamically adjust its thought process as needs be.

```
def ctm_loss(logits, targets):
    B, C, T = logits.shape
    # B=minibatch size, C=classes, T=thought steps
    # Targets shape: [B]
    # Compute certainties as 1 - normalised entropy
    p = F.softmax(logits, 1)
    log_p = torch.log_softmax(logits, 1)
    entropy = -torch.sum(p * log_p, dim=1)
    max_entropy = torch.log(C)
    certainties = 1 - (entropy / max_entropy)
    # Certainties shape: [B, T]
    # Expand targets over thought steps
    targets_exp = torch.repeat_interleave(targets.unsqueeze(-1), T, -1)
    # Loss function could be other things, but we use cross entropy without reduction
    loss_fn = nn.CrossEntropyLoss(reduction='none')
    # Losses are of shape [B, T]
    losses = loss_fn(predictions, targets_exp)
    # Get indices of lowest loss thought steps for each item in the minibatch
    lowest_idx = losses.argmin(-1)
    # Get indices of most certain steps for each item in the minibatch
    certain_idx = certainties.argmax(-1)
    loss = (losses[:, lowest_idx] + losses[:, certain_idx])/2
    return loss.mean()
```

Listing 4 | CTM loss function, enabling the CTM to be flexible regarding the number of internal ticks used for any given data point.

Introducing timing as a fundamental functional element of the CTM has a number of beneficial properties, one of which is that we can train the CTM **without any restriction on how many internal ticks it should use**. Such freedom, while subtle, is actually quite profound as it lets the CTM attribute variable amounts of compute to different data points. The idea of adaptive/dynamic compute (Graves, 2016) is aligned with modern test-time compute, with the difference being that this sought-after modeling property falls out of the CTM as a consequence, rather than it being applied post-hoc or as a restriction during learning. Note that there is nothing in the loss function that explicitly encourages

a找到相应的置信度度量 C^t。我们简单地将置信度计算为 1 - 归一化熵。
We 计算 $\mathcal{L}^t$ 和 C^t 对于所有 $t \in \{1, \ldots, T\}$，得到每个内部时间片的损失和置信度，$\mathcal{L} \in$ ^{T}
and $C \in$ ^{T}.

一个自然的问题是：我们应该如何将 $\mathcal{L}$ 减少为一个标量损失以进行学习？我们的损失函数旨在优化 CTM 在内部思维维度上的性能。我们不是依赖单一步骤（例如最后一个步骤），这可能会激励模型仅在该特定步骤输出，而是动态地从两个内部时钟中聚合信息：最小损失点和最大确定性点。这种方法有几个优势：（1）它鼓励 CTM 在多个内部时钟上发展有意义的表示和计算；（2）它自然地促进了阶梯式学习效果，模型可以最初利用较晚的内部时钟进行更复杂的处理，并逐渐过渡到较早的步骤进行更简单的处理；（3）它使 CTM 能够根据数据集中个别数据点的固有难度调整其计算。为此，我们动态地（按数据点）在两个内部时钟上聚合 $\mathcal{L}$：

1. 最小损失点：$t_1 = \text{argmin}(\mathcal{L})$；和 2. 最大确定性点
：$t_2 = \text{argmax}(\boldsymbol{C})$。

The final loss is then computed as: 终最损失然后计算为：

$$L = \frac{\mathcal{L}^{t_1} + \mathcal{L}^{t_2}}{2}, \tag{12}$$

and 随机梯度下降用于优化模型参数，θ_{syn} 和 $\theta_{d=1\ldots D}$ (see
E方程 1 和 3)。

这种方法有效地使CTM能够改进其‘最佳’预测，同时确保高确定性被赋予正确的预测。如列表4所示，损失是如何计算的，展示了确定性是如何计算为1-归一化熵的。这种方法还使CTM能够根据需要动态调整其思维过程。

```
def ctm_loss(logits, targets):
    B, C, T = logits.shape
    # B=minibatch size, C=classes, T=thought steps
    # Targets shape: [B]
    # Compute certainties as 1 - normalised entropy
    p = F.softmax(logits, 1)
    log_p = torch.log_softmax(logits, 1)
    entropy = -torch.sum(p * log_p, dim=1)
    max_entropy = torch.log(C)
    certainties = 1 - (entropy / max_entropy)
    # Certainties shape: [B, T]
    # Expand targets over thought steps
    targets_exp = torch.repeat_interleave(targets.unsqueeze(-1), T, -1)
    # Loss function could be other things, but we use cross entropy without reduction
    loss_fn = nn.CrossEntropyLoss(reduction='none')
    # Losses are of shape [B, T]
    losses = loss_fn(predictions, targets_exp)
    # Get indices of lowest loss thought steps for each item in the minibatch
    lowest_idx = losses.argmin(-1)
    # Get indices of most certain steps for each item in the minibatch
    certain_idx = certainties.argmax(-1)
    loss = (losses[:, lowest_idx] + losses[:, certain_idx])/2
    return loss.mean()
```

列表 4 | CTM 损失函数，使 CTM 能够灵活地根据任何给定数据点使用的内部时钟数目的多少进行调整。

引入时间作为CTM的基本功能元素具有许多有益的性质，其中之一是，我们可以训练CTM而不对其应使用多少内部节拍进行任何限制。这种自由虽然微妙，但实际上相当深刻，因为它使CTM能够为不同的数据点分配不同的计算量。自适应/动态计算（Graves, 2016）的概念与现代测试时计算相一致，不同之处在于，这种所需的建模性质是CTM的一个自然结果，而不是事后应用或在学习过程中作为限制。请注意，损失函数中没有任何内容明确鼓励

this behavior. In some sense, the CTM implements a form of *inductive bias*, where the complexity of the modeling process (approximated by the number of internal ticks used) can be tailored to the data. We believe that this is a far more natural means by which to solve problems of variable difficulty (e.g., easy versus difficult to classify images). That such a property occurs as a consequence of improving biological plausibility is not surprising, but it is gratifying. We contrast the performance and characteristics of the CTM against other models and a human baseline in Section 5.

Experimental evaluation

The following sections present a comprehensive evaluation of the Continuous Thought Machine (CTM) across a diverse suite of challenging tasks. The primary goal of these experiments is to explore the capabilities and characteristics that emerge from the CTM's core design principles: **neuron-level temporal processing** and the use of **neural synchronization as a direct latent representation**. We aim to understand how explicitly modeling and leveraging the unfolding of internal neural activity allows the CTM to approach problems requiring different facets of intelligence.

We begin by examining the CTM on standard perception tasks like ImageNet-1K (Section 3), focusing on analyzing the richness of its internal dynamics, its emergent reasoning processes, calibration properties, and adaptive computation. We supplement these later with studies to compare the CTM to human performance on CIFAR-10 (Section 5) and perform ablation studies on CIFAR-100 (Section 6).

We then specifically probe the CTM's capacity for complex sequential reasoning, planning, and spatial understanding using a challenging 2D maze navigation task designed to necessitate the formation of an internal world model (Section 4).

Further experiments investigate the CTM's ability to learn and execute algorithmic procedures on sequence-based tasks such as sorting real numbers (Section 7) and cumulative parity computation (Section 8), where the temporal unfolding of thought is critical. We also test its capacity for memory, retrieval, and symbolic manipulation through a question-answering task on MNIST digits (Section 9). Finally, we extend the CTM to reinforcement learning environments to demonstrate its applicability to sequential decision-making and continuous interaction with an external world (Section 10).

Collectively, these experiments are designed to provide insights into how grounding computation in neural dynamics allows the CTM to develop and utilize internal thought processes, offering a distinct approach compared to conventional models and taking a step towards more biologically plausible artificial intelligence.

3. ImageNet-1K classification

In this section we test the CTM on the ImageNet-1K classification task. We are not claiming that the CTM is state-of-the-art in terms of classification accuracy, which would require a substantial amount of effort and tuning to find an optimal training recipe (Vryniotis and Cord, 2021), but rather that the way in which the CTM solves this task is novel and worth inspection. We detail the model setup and hyperparameters in Appendix C.1. With a ResNet-152 backbone[3], the CTM achieves 72.47% top-1 validation accuracy and 89.89% top-5 validation accuracy, when assessed on uncropped ImageNet-1K validation data (but scaled such that the short side of the image is length 256). While this result is currently not comparable with state-of-the-art techniques, it is also the first attempt to classify ImageNet-1K using neural dynamics as a representation. We anticipate closing this gap that with further advances, more hyperparameter tuning, and feature extractors tailored to the CTM.

[3] altered such that the initial convolution is kernel is 3×3 as opposed to 7×7 to constrain the receptive field – see Appendix C.1 for a discussion thereon

这种行为。某种意义上，CTM 实现了一种形式的 *inductive bias*，其中建模过程的复杂性（通过内部时钟的数量近似表示）可以适应数据。我们认为，这是一种更为自然的方式来解决不同难度的问题（例如，容易分类与难以分类的图像）。这种性质之所以会在提高生物可行性的同时出现，这并不令人惊讶，但却是令人欣慰的。我们在第 5 节中将 CTM 的性能和特征与其他模型和人类基线进行对比。

实验评估

以下部分呈现了对连续思维机器（CTM）在一系列具有挑战性的任务中的全面评估。这些实验的主要目标是探索从CTM核心设计原则中涌现出来的能力和特征：神经元级时间处理以及使用神经同步作为直接潜在表示。我们旨在理解明确建模和利用内部神经活动的展开如何使CTM能够接近需要不同方面智能的问题。

我们首先在标准感知任务如ImageNet-1K（第3章）上检查CTM，重点关注其内部动力学的丰富性、涌现的推理过程、校准属性和适应性计算。随后我们在CIFAR-10上将CTM与人类性能进行比较（第5章），并在CIFAR-100上进行消融研究（第6章）。

We 然后具体探究CTM在复杂序列推理、规划和空间方面的能力
u理解使用一个具有挑战性的2D 迷宫导航任务，该任务设计旨在 necessitate 形成
a内部世界模型（第4节）。

进一步的实验调查了CTM在排序实数（第7节）和累积奇偶计算（第8节）等基于序列的任务上学习和执行算法性程序的能力，其中时间展开的思维至关重要。我们还通过MNIST数字的问答任务测试其记忆、检索和符号操作的能力（第9节）。最后，我们将CTM扩展到强化学习环境中，以展示其在序列决策和与外部世界持续交互中的适用性（第10节）。

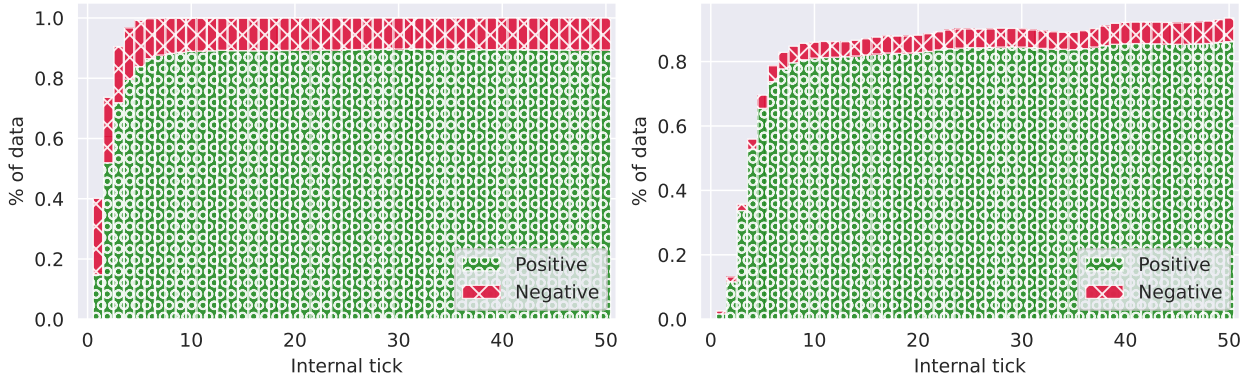
这些实验旨在提供关于将计算接地于神经动力学如何使CTM发展和利用内部思维过程的见解，这与传统模型提供了不同的方法，并朝着更加生物合乎情理的人工智能迈进。

3. ImageNet-1K分类

在本节中，我们测试了CTM在ImageNet-1K分类任务上的表现。我们并不声称CTM在分类准确性方面是最先进的，这需要大量的努力和调整来找到最佳的训练配方（Vryniotis和Cord, 2021），而是认为CTM解决此任务的方式是新颖的，值得检查。我们将在附录C.1中详细说明模型设置和超参数。使用ResNet-152骨干[3]，CTM在未裁剪的ImageNet-1K验证数据上（但图像的短边长度调整为256）实现了72.47%的top-1验证准确率和89.89%的top-5验证准确率。虽然这一结果目前尚无法与最先进的技术相比，但它也是首次尝试使用神经动力学作为表示来分类ImageNet-1K。我们期待随着进一步的发展，通过更多的超参数调整和针对CTM的特征提取器，能够缩小这一差距。

[3] altered such that the initial convolution is kernel is 3×3 as opposed to 7×7 to constrain the receptive field – see Appendix C.1 for a discussion thereon

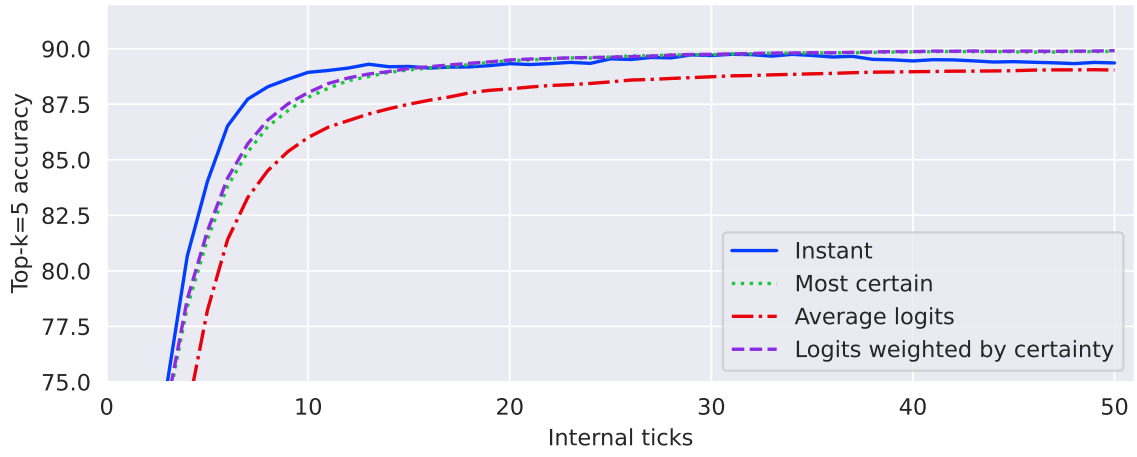
3.1. Prediction analysis: the power of the thought dimension



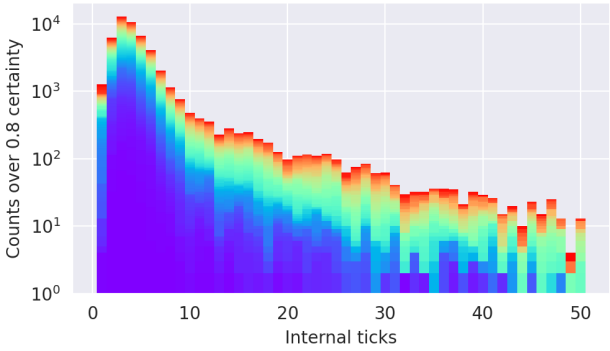
(a) Certainty threshold of 0.5.

(b) Certainty threshold of 0.8.

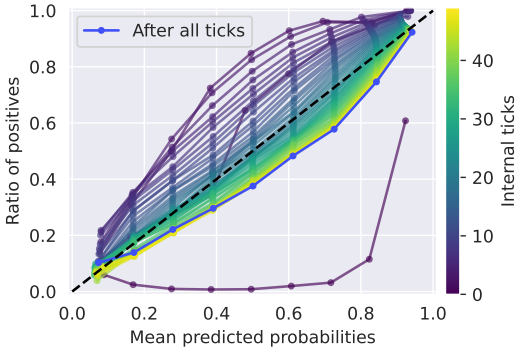
Figure 2 | Top-5 accuracies (validation) only when making a prediction past fixed certainty thresholds. At a lower threshold of (a) 0.5, the CTM will make a prediction 100 % of the time from approximately 4 internal ticks and onward, but (b) will only predict on approximately 80 % of the data when setting a certainty threshold of 0.8. These assessments could be used to tailor a certainty threshold to enable adaptive compute, halting compute as needed.



(a) Accuracy vs. steps for different prediction mechanisms.



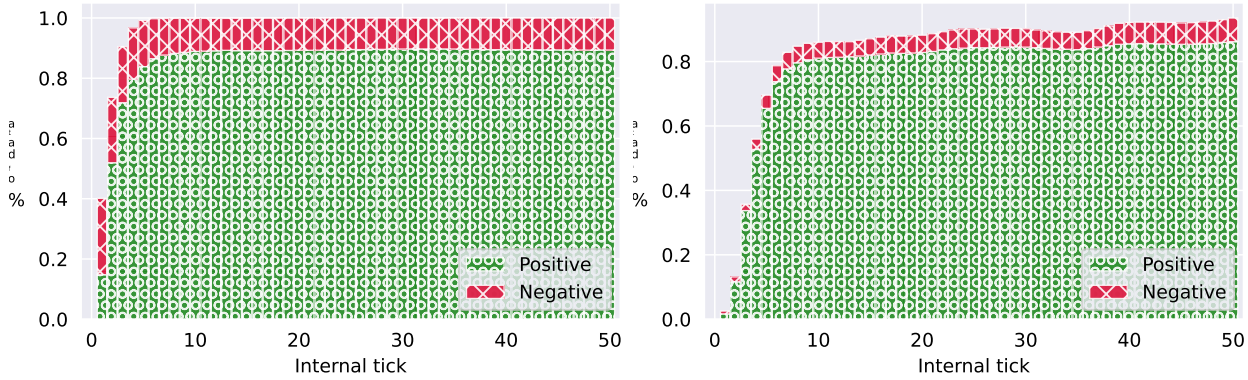
(b) Histogram of certainty of at least 0.8.



(c) Calibration plots.

Figure 3 | Exploration of the performance and utility of the CTM, showing the relationship between internal ticks and top-5 ImageNet-1K accuracy. In (a) we show the accuracy versus internal ticks when determining the output prediction in 4 different ways, showing how taking the prediction at a given internal tick is sensible until approximately 15 steps, where it becomes better to consider the certainty as a measure of success. In (b) we show a histogram of data counts exceeding a certainty of 0.8 for each internal tick; color denotes class indices. In (c) we show calibration plots, where predicted probabilities for the CTM are considered to be the average probability up to a given internal tick, showing how this results in good model calibration.

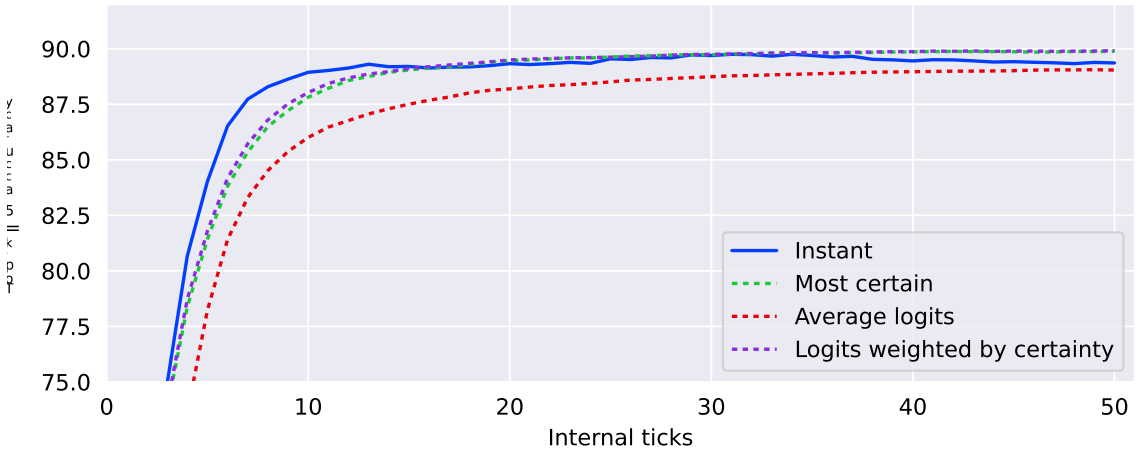
3.1. 预测分析：思维维度的力量



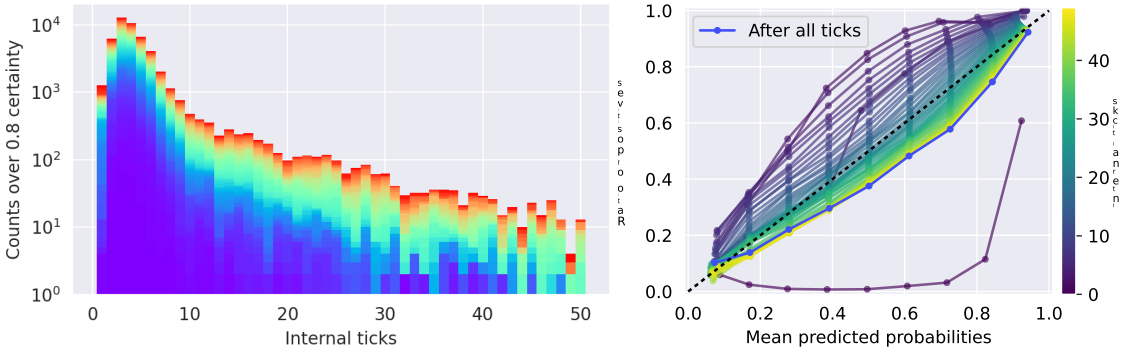
(a) Certainty threshold of 0.5.

(b) Certainty threshold of 0.8.

图 2 | 只在预测超过固定置信阈值时的 Top-5 准确率（验证）。在较低的阈值 (a) 0.5 时，CTM 从大约第 4 个内部时钟周期开始 100% 的时间进行预测，但在将置信阈值设置为 (b) 0.8 时，只在大约 80% 的数据上进行预测。这些评估可以用于调整置信阈值以实现适应性计算，根据需要停止计算。



(a) Accuracy vs. steps for different prediction mechanisms.



(b) Histogram of certainty of at least 0.8.

(c) Calibration plots.

图 3 | 探索 CTM 的性能和实用性，展示内部时钟与 top-5 ImageNet-1K 准确率之间的关系。在 (a) 中，我们展示了在四种不同的方式下确定输出预测时的准确率与内部时钟的关系，显示在大约 15 步之前，根据给定内部时钟取预测是合理的，而之后考虑确定性作为成功度量更好。在 (b) 中，我们展示了每个内部时钟超过 0.8 确定性的数据计数直方图；颜色表示类别索引。在 (c) 中，我们展示了校准图，其中认为 CTM 的预测概率是截至给定内部时钟的平均概率，显示这导致了良好的模型校准。

Figure 2 shows how the internal ticks for the CTM could be truncated when a chosen minimum certainty is reached, and what the expected top-5 accuracy would be. For instance, it takes fewer than 20 internal ticks per image to reach a certainty of 0.5 for all data, but when choosing a threshold of 0.8, not all of the data always reaches this threshold. In the latter case, compute could be truncated as needed by the user. By halting the internal ticks when an acceptable internal threshold is met, a form of adaptive compute can be utilized.

Prediction mechanisms: accounting for certainty. Figure 3a shows how different prediction mechanisms can affect overall performance. We show an 'instant' prediction (i.e., the prediction at each internal tick) compared to predictions based on certainty: the maximum certainty up to a given step, and when weighting logits by certainty. Interestingly, after approximately 15 internal ticks it becomes preferable to take explicit account of certainty. Using an unweighted average of logits yields the worst performance, implying that the CTM does indeed undergo a process to improve its prediction, while *potentially moving through incorrect predictions of low certainty*. Figure 5b gives concrete examples of low-certainty instances for further evidence.

Figure 3b shows the distribution of internal ticks when the CTM yields a certainty of at least 0.8, indicating that the majority of the data requires fewer than 10 internal ticks, with a long tail toward a maximum of 50 internal ticks. The calibration plots in Figure 3c are perhaps most striking as they show that the CTM has very good calibration. This is owing to how the CTM becomes increasingly certain over internal ticks: we consider the predicted probability of a given instance to be the average probability over internal ticks of the chosen class. The demonstration in Figure 5 shows how certainty increases over internal ticks. Evidently, following an internal process seems to enable the CTM to produce more trustworthy class probabilities – a characteristic that usually needs post-training adjustments or special training setups (Guo et al., 2017).

3.2. Neural dynamics analysis

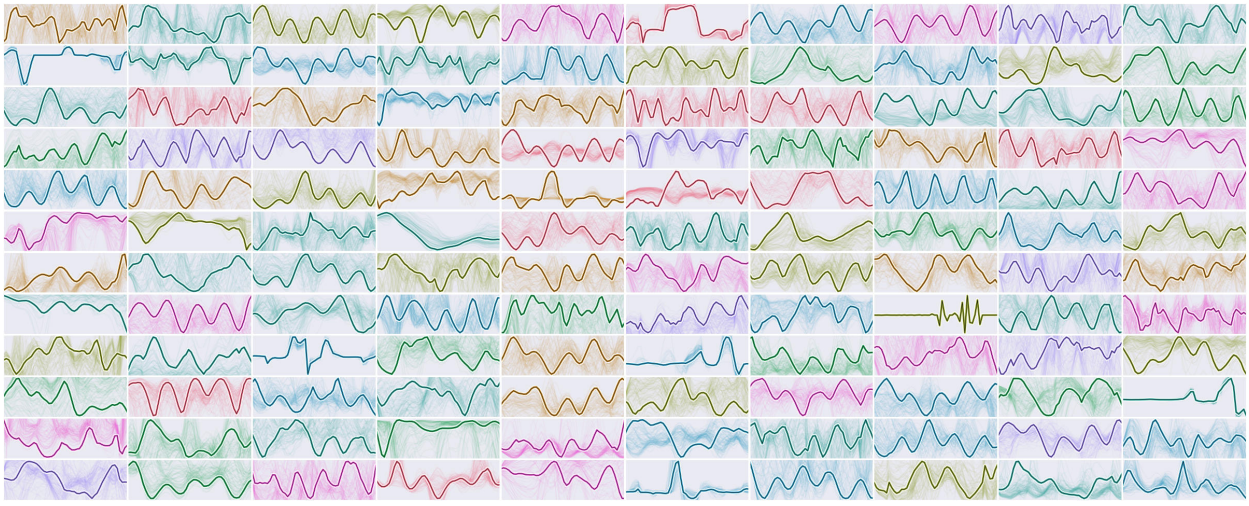


Figure 4 | Post-activation neuron dynamics (⑤ in Figure 1). Each subplot (in a random color) shows the dynamics of a single neuron over internal ticks, where multiple examples from different images are shown as faint background lines and the foreground line is a single example. We show multiple examples as evidence of diversity across data. It is these dynamics that are used when computing synchronization and they form the fundamental processing element of the CTM.

Figure 4 visualizes the post-activation neural dynamics of this CTM. **These dynamics are diverse and rich in structure, and they form the representation with which the CTM takes action and makes decisions.** The purpose of this figure is to show that the CTM does indeed yield a diverse set of neural activities, the dynamics of which can be measured in relationship to one another (i.e.,

图 2 显示了当达到选定的最小置信度时 CTM 的内部刻度可能被截断的情况，以及预期的 top-5 准确率会是多少。例如，对于所有数据来说，每张图像只需要不到 20 个内部刻度就能达到 0.5 的置信度，但选择 0.8 的阈值时，并非所有数据都能达到这个阈值。在后一种情况下，计算可以由用户按需截断。通过在达到可接受的内部阈值时停止内部刻度，可以利用一种自适应计算的形式。

预测机制：考虑确定性。图3a展示了不同预测机制如何影响整体性能。我们展示了“即时”预测（即每次内部时钟的预测）与基于确定性的预测：给定步骤的最大确定性，以及按确定性加权的logits。有趣的是，在大约15个内部时钟之后，考虑确定性变得更有利。未加权的logits平均值导致最差的性能，这表明CTM确实经历了一个过程来改进其预测，而
potentially moving through incorrect predictions of low certainty。图5b提供了低确定性实例的具体例子，以进一步证明这一点。

图3b显示了当CTM的确定性至少为0.8时内部跳变的分布情况，表明大多数数据只需要少于10个内部跳变，但有一个长尾分布，最大值可达50个内部跳变。图3c中的校准图可能最为引人注目，因为它们显示CTM具有非常好的校准性。这是由于CTM在内部跳变过程中逐渐变得更加确定：我们考虑给定实例的预测概率是所选类别的内部跳变概率的平均值。图5中的演示显示了确定性如何随着内部跳变而增加。显然，遵循内部过程似乎使CTM能够生成更可信的类概率——这一特性通常需要在训练后进行调整或特殊的训练设置（Guo等，2017）。

3.2. 神经动力学分析

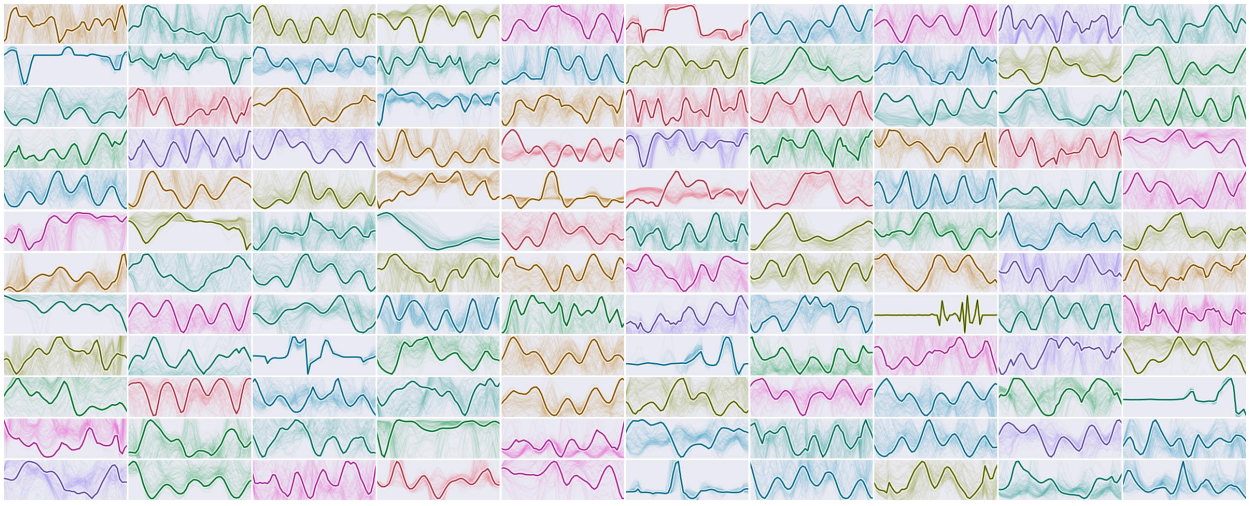


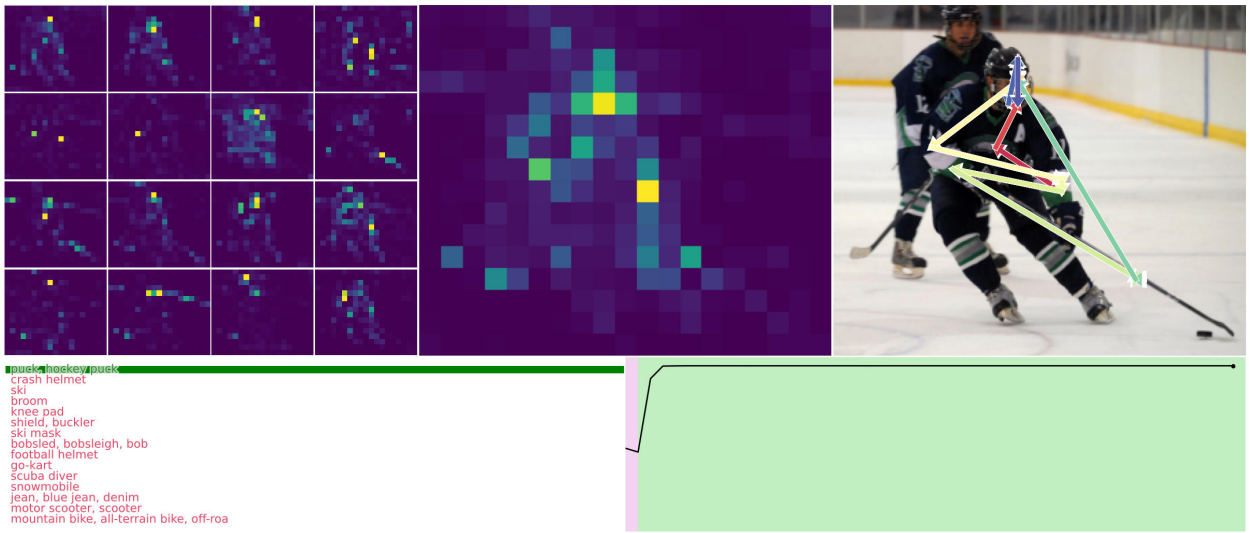
图 4 | 激活后的神经元动力学（在图 1 中）。每个子图（随机颜色）显示了一个神经元在内部时钟上的动力学变化，其中来自不同图像的多个示例以淡背景线的形式展示，而前景线则是一个单独的示例。我们展示多个示例以证明数据之间的多样性。正是这些动力学在计算同步时被使用，并构成了 CTM 的基本处理单元。

图 4 展示了此 CTM 的后激活神经动力学。这些动力学是多样的且结构丰富，并形成了 CTM 采取行动和做出决策的表示。该图的目的是展示 CTM 确实产生了多样化的神经活动，这些活动的动力学可以相互关联来测量（即，

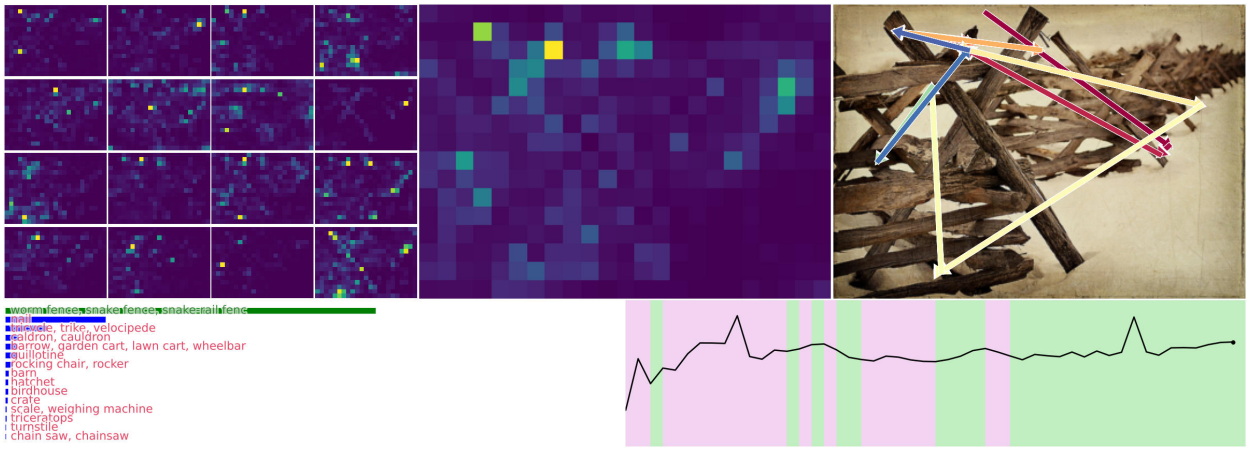
synchronization) and used as a powerful latent representation for downstream tasks. In Section 4 we provide evidence that such a representation has high-utility for problem solving.

Take-home message. Figure 4 shows that the neurons in the CTM exhibit complex multiscale patterns, but have not provided any pragmatic rationale for why this might be useful. The reason we show this is as evidence that we the CTM builds and leverages true **dynamics**, where the patterns of neural activity are non-trivial and diverse. These dynamics and the complexity contained within form a new kind of representation that we believe is closer to the biologically plausible mechanisms behind neural computation.

3.3. Demonstrations: the CTM follows a process



(a) A high-certainty example.



(b) A low certainty example that eventually becomes correct with more internal ticks.

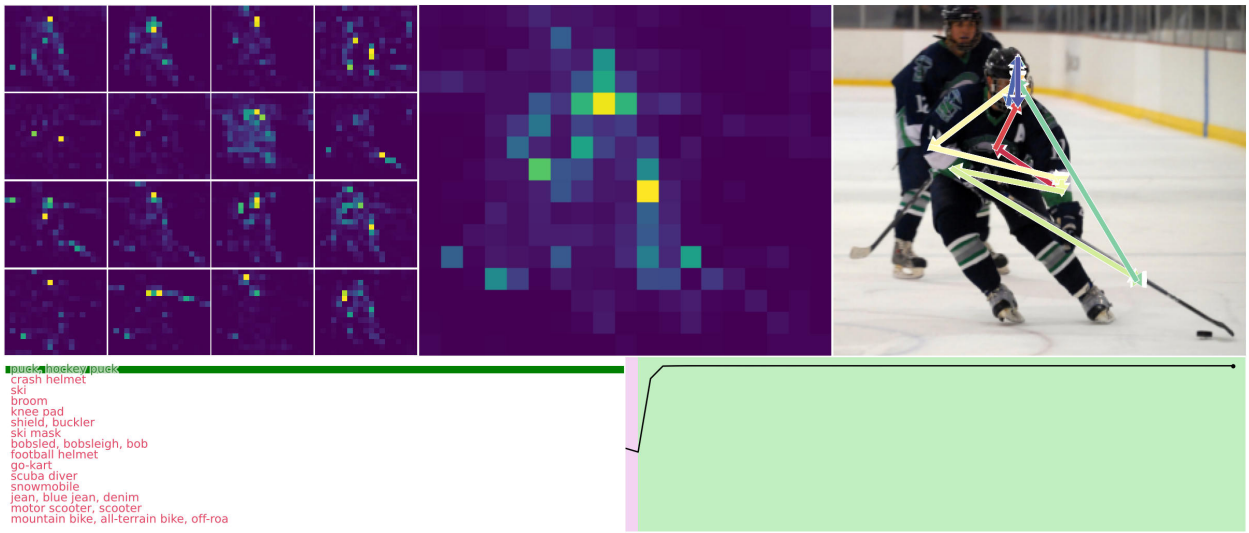
Figure 5 | ImageNet-1K use cases, randomly drawn from the validation set. This demonstration is ideal when viewed as a video as there are 50 internal ticks, but we are showing only the final step. On the left we show the average (over internal ticks) of all 16 attention heads' weightings, and on the right we show an approximation of the center of mass of their collective average (see Appendix C.1 for details) over the full 50 steps as arrows from red to blue. Owing to the sequential nature of the CTM's internal ticks, we observe each head's attention shifting smoothly from region to region, at times zoning into specific salient features (nose, boundaries, etc.), while at other times spreading over wider areas, or even moving in identifiable directions (e.g., from bottom to top). Appendix C.3 contains several additional demonstrations.

Figure 5 shows examples from the ImageNet-1K validation set, as the CTM sees it. We give more examples in Appendix C.3. We encourage the reader to view these visualizations in video form as there

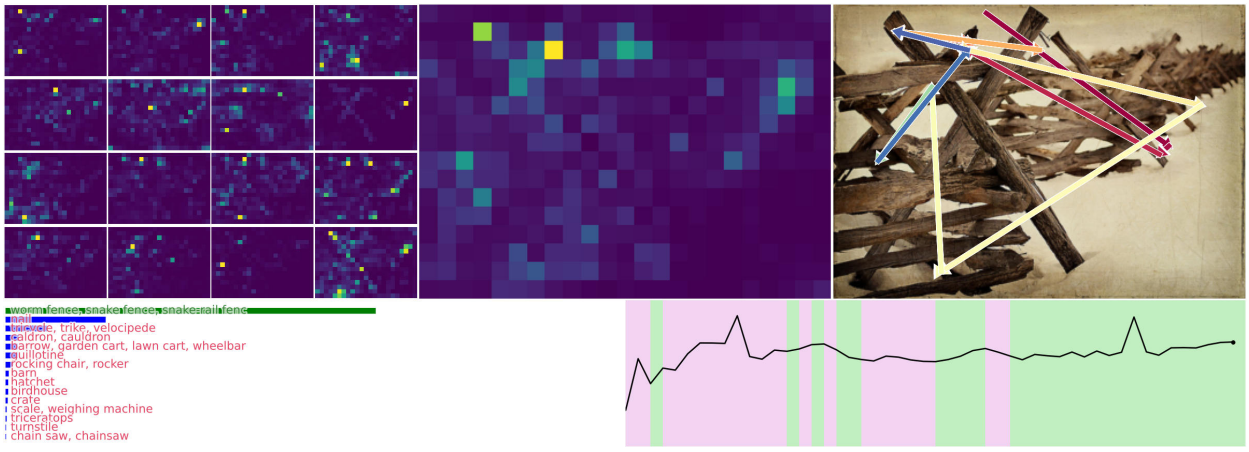
s同步) 并作为下游任务的强大潜在表示。在第4节中我们
p提供证据表明这种表示对于问题解决具有高实用性。

Take-home message. 图4显示CTM中的神经元表现出复杂的多尺度模式，但尚未提供任何实用的理由说明这为何有用。我们展示这一点是为了作为证据，证明CTM构建并利用了真正的动力学，其中神经活动的模式是非平凡且多样的。这些动力学及其包含的复杂性形成了我们相信更接近生物上合理的大脑计算机制的新类型表示。

3.3. 展示：CTM 遵循一个过程



(a) A high-certainty example.



(b) A low certainty example that eventually becomes correct with more internal ticks.

图 5 | ImageNet-1K 的应用场景，随机抽取自验证集。当将其视为视频时，此演示是理想的，因为共有 50 个内部刻度，但我们仅展示了最终步骤。在左侧，我们展示了所有 16 个注意力头的加权平均值（基于内部刻度），而在右侧，我们展示了它们集体平均值的质心近似（详见附录 C.1），以从红色到蓝色的箭头表示整个 50 步的过程。由于 CTM 内部刻度的顺序性质，我们观察到每个头的注意力平滑地从一个区域转移到另一个区域，在某些时候聚焦于特定的显著特征（如鼻子、边界等），而在其他时候则扩散到更广泛的区域，甚至在可识别的方向上移动（例如，从底部到顶部）。附录 C.3 包含了更多的演示。

F图5显示了CTM在其中看到的ImageNet-1K验证集的一些示例。我们提供更多
e附录C.3中的例子。我们鼓励读者以视频形式查看这些可视化内容，因为

are 50 frames (1 per internal tick) that show how **the attention maps change over time**, shifting to different regions over the CTM's internal thought process. The smooth transition of attention to various parts of the image emerges as a property during training. We attempted to show some of this transitory attention by way of arrows, showing how the attention moves over salient areas an intuitive fashion. Unpacking every interesting facet of these attention map progressions is simply infeasible in this paper. Instead, for these examples we show how the attention patterns demonstrate a complex process. Also shown is the certainty over time, indicating how the CTM becomes more certain as it reasons.

Note that since the CTM uses attention to retrieve information, it is not limited to fixed-size images (and, for future work, can be applied to arbitrarily length token sequences), hence our evaluations on uncropped validation data. One could conceivably build a **hierarchy of tokens** by using multiple input resolutions, letting the CTM attend to a large collection of tokens (during inference, not training), but we reserve this exploration for future work.

The CTM learns to observe over time. During our experimentation we monitored the functionality of the CTM as it progressed through its training. While we do not explicitly show it in this paper, the complexity of the neural dynamics and consequently the complexity of the observation process the CTM undertakes increases during learning. At first the CTM does not 'look around' as it does in Figure 5, only learning that behavior over time. An early work by Xu et al. (2015) demonstrated how to use an RNN to reason over an image for text captioning; the CTM's reasoning process differs in that it unfolds along an internal dimension that is decoupled from both the input and target data, yet it still yields a complex attention pattern that highlights where it focuses its attention when making decisions.

Taking steps toward natural intelligence. Biological intelligence is still superior to AI in many cases (Chollet et al., 2024; Lake et al., 2017; Phan et al., 2025; Ren and Xia, 2024). Biological brains solve tasks very differently to conventional neural networks, which might explain why this is the case. In this work, we aimed to develop a model that approaches problem-solving in a manner more aligned with biological brains, emphasizing the central role of neural dynamics in achieving this similarity. Our observations suggest that the CTM undertakes a process, where it sequentially retrieves information from an image. We show more examples in Appendix C.3, where each instance shows interesting or unique patterns or outcomes.

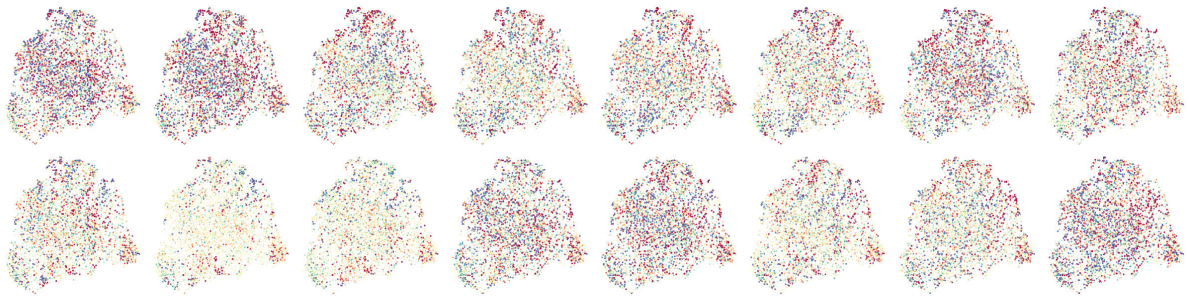


Figure 6 | Observation of the neurons in the CTM as it observes and thinks about an image. Appendix J gives details of how the neurons were arrange using UMAP (McInnes et al., 2018). The colors indicate activations that range from low (blue) to high (red). We show the progression of the neural activity over internal ticks from top left to bottom right. Upon careful inspection one can discover clear structures at multiple scales. This visualization is best viewed in video form.

As a final point of comparison, we consider low-frequency traveling waves, a phenomenon widely documented in cortical dynamics and implicated in various neural computations (Muller et al., 2018). We present Figure 6, where we map the CTM's neurons to a 2D feature space using UMAP (McInnes

有50帧（每帧对应一次内部时间步）展示了注意力图随时间的变化，随着CTM的内部思考过程，注意力会转移到不同的区域。注意力平滑地转移到图像的不同部分，这一特性在训练过程中逐渐显现。我们尝试通过箭头来展示部分过渡中的注意力，以直观的方式显示注意力如何移动到显著区域。这些注意力图进化的每一个有趣方面在这篇文章中都难以完全展开。相反，对于这些示例，我们展示了注意力模式如何体现一个复杂的过程。同时，还展示了随着时间的推移，CTM的确定性增加，表明它在推理过程中变得越来越确定。

注意，由于CTM使用注意力机制检索信息，它不限于固定大小的图像（并且在未来的工作中，可以应用于任意长度的标记序列），因此我们在未裁剪的验证数据上进行了评估。理论上可以通过使用多种输入分辨率构建标记的层次结构，让CTM在推理时（而非训练时）关注大量标记，但我们保留这项探索在未来的工作中进行。

CTM 学会在时间中观察。在我们的实验中，我们监控了 CTM 在训练过程中功能的变化。虽然我们在这篇论文中没有明确展示这一点，但随着学习的进行，神经动力学的复杂性增加，相应地，CTM 所进行的观察过程的复杂性也增加。最初，CTM 并不会像图 5 中那样“环顾四周”，而是在一段时间后才学会这种行为。徐等人（2015）的一项早期工作展示了如何使用 RNN 对图像进行推理以生成文本描述；而 CTM 的推理过程有所不同，它沿着一个与输入数据和目标数据均解耦的内部维度展开，但仍会产生一个复杂的注意力模式，突出显示它在做决策时的注意力焦点。

朝着自然智能迈进。生物智能在许多情况下仍然优于AI（Chollet等，2024；Lake等，2017；Phan等，2025；Ren和Xia，2024）。生物大脑在解决任务的方式与传统神经网络非常不同，这可能就是这种情况的原因。在本工作中，我们旨在开发一种模型，使其在解决问题的方式上更接近生物大脑，强调神经动力学在实现这一相似性中的核心作用。我们的观察表明，CTM 采取一个过程，其中它顺序地从图像中检索信息。我们在附录C.3中提供了更多的例子，其中每个实例都展示了有趣或独特的模式或结果。

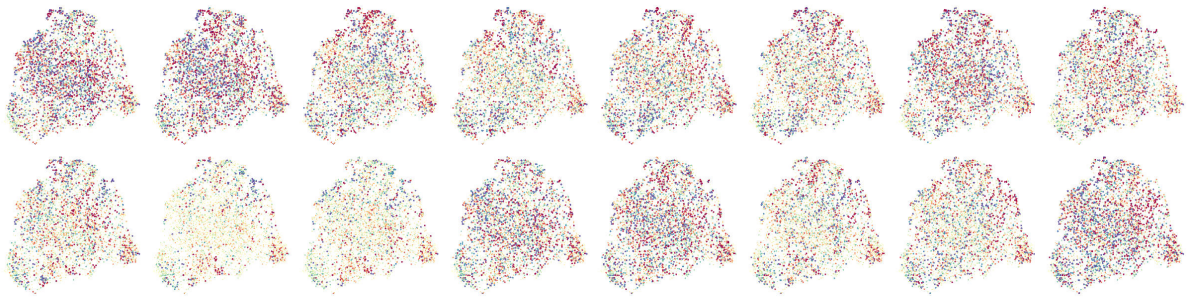


图 6 | CTM 在观察和思考图像时的神经元观察。附录 J 提供了使用 UMAP（McInnes 等，2018）排列神经元的详细信息。颜色表示从低（蓝色）到高的激活程度。我们从左上角到右下角展示了神经活动的进展。仔细检查后，可以发现多个尺度上的清晰结构。这种可视化最好以视频形式查看。

A作为最后一点比较，我们考虑低频行波，这是一种广泛存在的现象
d记录在皮层动力学中，并与各种神经计算相关（Muller等，2018年）。
We 现在呈现 Figure 6，其中我们使用 UMAP（McInnes）将 CTM 的神经元映射到 2D 特征空间。

et al., 2018). Each neuron's position in this space is determined by its activation 'profile' – its response pattern over both time and multiple stimuli (see Appendix J). Visualizing this mapping over internal ticks reveals low-frequency structures propagating across the feature space (best viewed as a video). Importantly, the CTM generates this structure in an emergent fashion, without any explicit driving signal. Analogous phenomena occur in networks of Kuramoto oscillators (Miyato et al., 2024); in our case, waves propagate across a learned feature map in an all-to-all network. Concurrent work also explores explicitly encoding traveling waves for long-range communication (Jacobs et al., 2025). We do not assign functional meaning to these observed waves but highlight their distinct presence during the CTM's thought process.

4. 2D Mazes: a setup that requires complex sequential reasoning

In this section, we use 2D mazes as a tool to investigate the behavior of the CTM when asked to plan and navigate. Solving a 2D maze can be easy with the right inductive bias: by ensuring the output space matches the dimensions of the input space, where at each pixel a model must perform binary classification. Such a setup is amenable to machines by design, as they can learn iterative algorithmic solutions (Bansal et al., 2022; Schwarzschild et al., 2021), and it excludes the need to think in a more natural fashion. Even so, the trainability of models is questionable, with techniques often relying on careful model and/or objective design that favors generalization to larger mazes (Bansal et al., 2022; Zhang et al., 2025). Such generalization is certainly one important aspect of intelligence.

However, there is a critical distinction between simply finding the solution to a maze, versus **following a thought process to form said solution**. While the emergent behavior of such systems can be impressive (e.g., generalizing to mazes far greater in size (Bansal et al., 2022)), it is difficult to reconcile whether these models are demonstrating intelligence. How can we make the 2D maze task more challenging, such that a human-like solution is required? We propose the following:

1. **Constrain the output space directly** as a set of steps from start (denoted by a red pixel) to finish (green pixel). We require a solution in the form of a fixed-sized array (length 100[4]) containing 1 of 5 movement types (**L**eft, **R**ight, **U**p, **D**own, or **W**ait[5]) per step. This mitigates against the types of simple solutions described above and requires more of an understanding of the target maze.
2. **Disallow positional embeddings** when using attention. There are two reasons for this: (1) it forces the model to build an internal 'world representation', where it can only craft attention queries based on its ongoing understanding of the data; and (2) it enables seamless scaling to bigger maze images (see Section 4.3).

It is our hope that this new phrasing of the 2D maze task provides a challenging benchmark that can highlight models that are able to follow a thought process. We used the maze-dataset repository (Ivanitskiy et al., 2023) to generate 39×39 mazes for training and 99×99 for generalization tests[6]. We trained three model variants for comparison:

1. A **CTM** with a constrained ResNet-34 backbone, using the first two hyper-blocks only. This CTM is nearly identical in structure to those used in image classification (Sections 3). At each internal tick the CTM outputs a matrix that defines a route from the start location, $\mathbf{y}^t \in \mathbb{R}^{100\times5}$ (see Equation 6). It is trained with the variable-internal tick loss (Equation 12). We also adjust the loss function to use a curriculum approach, optimizing for early steps in the route over later steps. Appendix D.2 details hyper-parameters and Appendix D.3 explains the curriculum approach.

[4]This may be shorter than required for some mazes, in which case we ignore later steps.
[5]We use 'wait' classes for instances where the route is shorter than 100, and fill the target vector with wait classes
[6]See the CTM code repository

etal., 2018)。每个神经元在这个空间中的位置由其激活‘配置文件’决定——即其在时间和多种刺激上的反应模式（参见附录J）。通过内部时间刻度可视化这种映射，可以揭示低频结构在特征空间中传播（最好以视频形式查看）。重要的是，CTM 是以一种涌现的方式生成这种结构，而没有任何明确的驱动信号。类似的现象也发生在库拉莫托振子网络中（Miyato et al., 2024）；在我们的情况下，波在全连接的特征图中传播。同时进行的研究还探索了显式编码行波以实现长距离通信（Jacobs et al., 2025）。我们不对观察到的这些波赋予功能意义，但强调它们在 CTM 的思维过程中具有独特的存在。

4. 二维迷宫：一种需要复杂序列推理的设置

在本节中，我们使用2D 迷宫作为工具来研究CTM 在被要求规划和导航时的行为。用合适的归纳偏置来解决2D 迷宫可以很简单：通过确保输出空间与输入空间的维度匹配，在每个像素上模型必须执行二元分类。这样的设置从设计上就适合机器，因为它们可以学习迭代的算法解决方案（Bansal 等人，2022；Schwarzschild 等人，2021），并且它排除了需要以更自然的方式思考的需求。即使如此，模型的可训练性仍存疑，技术往往依赖于精心设计的模型和/或目标，以有利于对更大迷宫的泛化（Bansal 等人，2022；Zhang 等人，2025）。这种泛化无疑是智能的一个重要方面。

然而，找到迷宫的解决方案和遵循思维过程形成该解决方案之间有一个关键区别。虽然此类系统的涌现行为可能令人印象深刻（例如，泛化到规模大得多的迷宫（Bansal等，2022）），但很难确定这些模型是否展示了智能。我们如何使2D迷宫任务更具挑战性，以要求类似人类的解决方案？我们提出以下方法：

1. 直接将输出空间限制为从起始点（用红色像素表示）到终点（绿色像素）的一系列步骤。我们要求解决方案以固定大小的数组（长度100[4]）的形式给出，其中包含每步1种移动类型（左、右、上、下或等待[5]）。这避免了上述简单解决方案，并要求对目标迷宫有更深入的理解。2. 在使用注意力机制时禁止使用位置嵌入。这样做的两个原因分别是：(1) 这迫使模型构建一个内部的“世界表示”，它只能根据对数据的持续理解来构建注意力查询；(2) 这使得模型能够无缝扩展到更大的迷宫图像（参见第4.3节）。

我们希望这种新的2D迷宫任务表述能够提供一个具有挑战性的基准，以突出那些能够遵循思考过程的模型。我们使用了maze-dataset仓库（Ivanitskiy等，2023）生成了39×39个迷宫用于训练，以及99×99个迷宫用于泛化测试[6]。我们训练了三种模型变体进行比较：

1. 一个受限的ResNet-34主干的CTM，仅使用前两个超块。这种CTM在结构上几乎与图像分类中使用的CTM（第3节）相似。在每个内部时间戳，CTM输出一个矩阵，定义从起始位置$y^t \in$到$^{100\times5}$的路径，参见方程6)。它使用变量内部时间戳损失（方程12）进行训练。我们还调整了损失函数以使用课程学习方法，在路径的早期步骤上进行优化而不是后期步骤。附录D.2详细说明了超参数，附录D.3解释了课程学习方法。

[4]This may be shorter than required for some mazes, in which case we ignore later steps.
[5]We use ‘wait’ classes for instances where the route is shorter than 100, and fill the target vector with wait classes
[6]See the CTM code repository

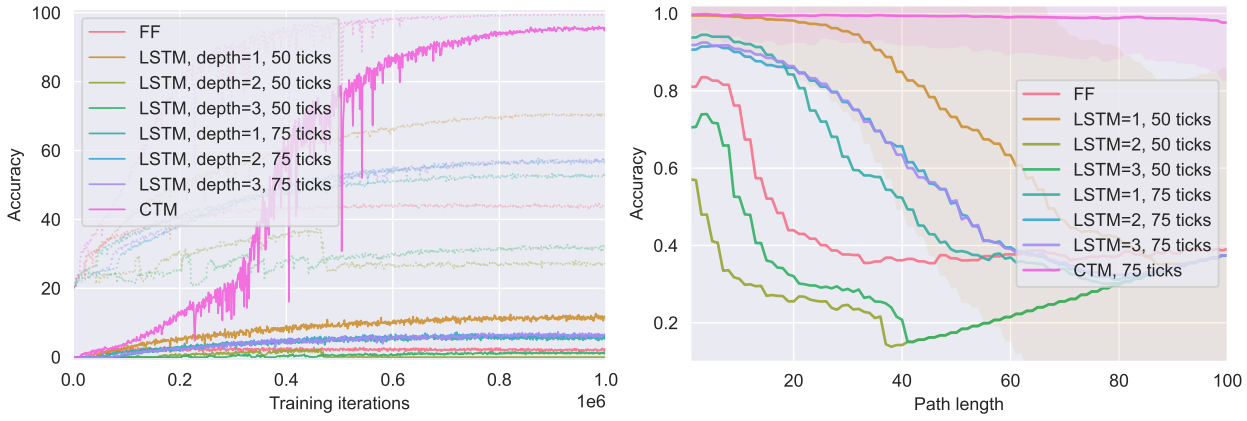
2. 1, 2, and 3-layer **LSTM baselines** using the same model width as the CTM (see Appendix D.4 for details). The LSTM baseline also uses a curriculum approach, but was unable to learn beyond a small number of steps in the maze.
3. A **feed-forward-only** (i.e., no recurrence) model (FF) where the features were projected via a hidden layer (same width as the CTM) to the prediction (see Appendix D.4 for details). Since we use no positional embedding for the CTM and LSTM models they must learn to build an internal representation of the data they are seeing, but no such mechanism is available to the FF model. Therefore, we flatten the final ResNet features and project those to $\mathbf{y}^t$ instead as this lets the FF model learn spatial context directly.

Using the same hidden width, the CTM required the fewest parameters. See Appendix D.4 for more details.

4.1. Results

Figure 7a shows the accuracies of the CTM versus baselines. The FF model and the best LSTM model both show signs of overfitting (see Appendix D.5 for loss curves), indicating that their structure is poorly suited to the problem. Only the CTM achieves high accuracy on this task. During our experimentation we simply could not get the LSTM to achieve the same performance, with the single-layer LSTM using 50 internal ticks achieving the best performance. Upon inspection of the solution, and as shown by the orange curve ("LSTM=1, 50 ticks") in Figure 7b, we can see that the LSTM is beginning to learn a solution, but is unable to push beyond that.

Trainability. The large disparity in performance between the CTM and LSTM in this case raises questions of trainability, with the CTM being far easier to optimize. Solving the maze task is complex because it requires a model to create and complex representation that modulates data interaction, produces a route prediction, and also maintains a memory of its positioning thus far (see Section 4.4 for a longer discussion). The fact that the CTM can do this with minimal modifications (only the form of the predictions) is testament to its utility.



(a) Accuracies, route (solid) and per step (dotted).

(b) Accuracies versus path length..

Figure 7 | CTM versus a feed-forward baseline and several LSTM setups. The CTM is the only model that can fit sufficiently to the training data (nearly perfect train accuracy), does not overfit (indicating a good choice of inductive bias (Utgoff, 2012)), and achieves high test accuracy for longer paths.

Figure 7b shows accuracies versus route length on the held-out test set. The CTM is clearly more capable of solving longer mazes, while the baseline methods start failing early on, with the best performing LSTM losing capability after approximately 20 steps along maze paths. What this indicates is that the CTM is more capable of learning to solve difficult problems. The depth 1 LSTMs were the

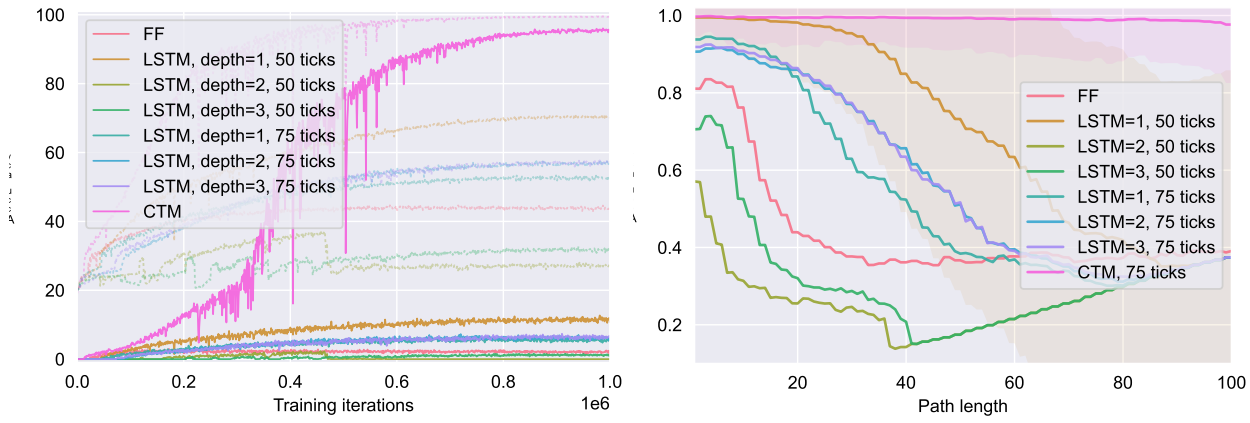
2. 使用与CTM相同模型宽度的1层、2层和3层LSTM基线（详情参见附录D.4）。LSTM基线也使用了渐进式学习方法，但在迷宫中仅能学习到少量步骤。

3. 一种仅前馈（即，无递归）的模型（FF），其中特征通过一个隐藏层（与CTM相同宽度）投影到预测中（详细内容参见附录D.4）。由于我们未为CTM和LSTM模型使用位置嵌入，因此它们必须学会构建它们所看到的数据的内部表示，但FF模型没有这样的机制。因此，我们将最终的ResNet特征展平并投影到y^t，这样可以让FF模型直接学习空间上下文。

U使用相同的隐藏层宽度，CTM所需的参数最少。更多内容请参见附录D.4。
d细节。

4.1. 结果
图7a显示了CTM与基线模型的准确率。FF模型和最佳LSTM模型都显示出过拟合的迹象（参见附录D.5中的损失曲线），表明它们的结构不适用于该问题。只有CTM在该任务上实现了高准确率。在我们的实验中，我们根本无法让LSTM达到相同的性能，单层LSTM使用50个内部时钟时表现最佳。通过对解决方案的检查，如图7b中的橙色曲线（“LSTM=1, 50个时钟”）所示，我们可以看到LSTM开始学习解决方案，但无法超越这一点。

Trainability. 在这种情况下，CTM和LSTM之间的性能差异引发了关于可训练性的疑问，CTM明显更容易优化。解决迷宫任务是复杂的，因为它要求模型创建和复杂表示以调节数据交互、生成路径预测，并且还要保持其迄今为止位置的记忆（详见第4.4节的更长讨论）。CTM仅通过修改预测的形式就能做到这一点，这证明了它的实用性。



(a) Accuracies, route (solid) and per step (dotted).

(b) Accuracies versus path length..

F图7 | CTM与前馈基线以及几种LSTM设置进行比较。只有CTM模型能够充分拟合
to 训练数据（几乎完美的训练准确率），不会过拟合（表明一个好的先入之见选择（Utgoff, 2012)), 并且在较长路径上实现了高测试准确率。

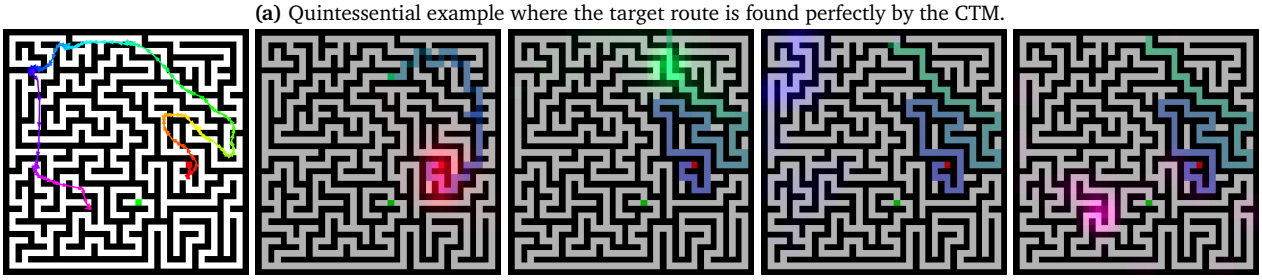
Figure 7b 显示了在保留测试集上的准确率与路径长度的关系。CTM 明显更能解决更长的迷宫，而基线方法则在早期就开始失效，表现最好的 LSTM 在大约 20 步后也开始失去能力。这表明 CTM 更擅长学习解决复杂问题。深度为 1 的 LSTM 是

closest in terms of parameter counts, but all baselines had more parameters. In other words, the CTM is not performing better because it has more parameters, but rather because of the ideas it is based upon: **neural dynamics and synchronization are useful**.

4.2. Demonstrations: the CTM learns the general procedure

Figure 8 shows the process followed by the CTM. By visualizing the average (across heads) attention weights over time, we can see how the CTM methodically steps along a plausible path until it reaches what it predicts to be the end of the maze. This problem-solving process is *not quite entirely unlike* how a human might approach solving a maze from the top down. We remind the reader to note now that this maze-solving CTM is not using any positional embedding, meaning that in order for it to follow a path through the maze it must craft the cross-attention query by 'imagining' the future state of the maze: a process known as 'episodic future thinking' (Atance and O'Neill, 2001) in humans.

(a) Quintessential example where the target route is found perfectly by the CTM.



(b) An example of **emergent behavior**, where the CTM attends along the path beyond its training steps.

Figure 8 | A natural maze solving approach. Each row shows how the CTM solves a different 39×39 maze. The leftmost images show, as colored arrows, the center of mass of attention throughout the thought processes, demonstrating how the CTM attends along the solution route. The images to the right are snapshots of the solution the CTM is outputting at different internal ticks, overlayed with an attention heatmap (color matched to the arrows). (a) shows a quintessential example, and (b) shows how the CTM continues to attend along the route even beyond the internal ticks used during training (for this we let the CTM unfold $2\times$ longer than it was trained for). We give additional examples and provide an **interactive demonstration where you can interact with the CTM** to solve mazes like this on our project page.

This CTM was trained to predict up to **100 steps outward from the start location**. In Figure 8a we can see how the CTM fixates its attention on the end of the maze (rightmost green heatmap; approximately 75 internal ticks) since this is within the 100 steps it can predict. Contrast this with Figure 8b, where the ground-truth path is far longer than 100 steps. The attention pattern **continues to trace out the remainder of the path** when we assess the attention maps using more internal ticks than what was used during training: to produce these visualizations we ran the CTM for $2\times$ the internal ticks it was trained on.

This behavior is emergent and suggests that the CTM has learned a general procedure for the underlying maze task, as opposed to merely memorizing the training data.

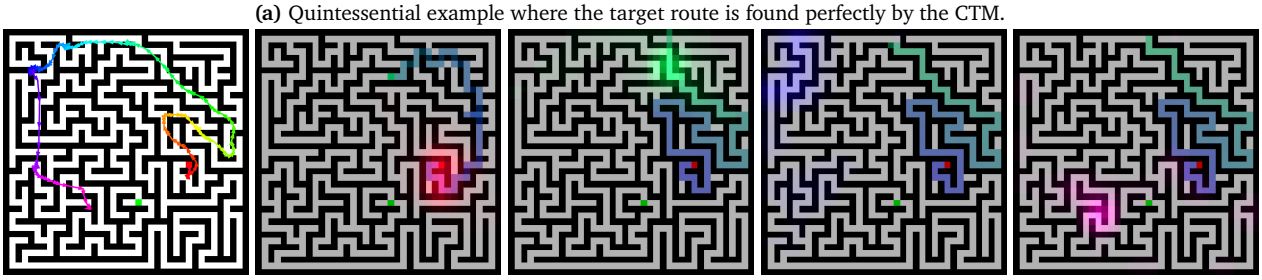
In the following section we demonstrate how this CTM can generalize to mazes bigger than what it was trained on.

c在参数量上最少，但所有基线模型的参数量更多。换句话说，CTM
is 并不是因为参数更多而表现更好，而是因为它基于的一些想法
upon: 神经动力学和同步是有用的。

4.2. 展示：CTM 学习通用程序

图8显示了CTM遵循的过程。通过可视化每个头的平均注意力权重随时间的变化，我们可以看到CTM如何有条不紊地沿着一条合理的路径行进，直到它预测到迷宫的终点。这一解决问题的过程类似于人类从上到下解决迷宫的方式。我们提醒读者注意，这个迷宫解决的CTM并没有使用任何位置嵌入，这意味着为了能够通过迷宫，它必须通过“想象”迷宫的未来状态来构建交叉注意力查询：这一过程在人类中被称为“情景未来思考”（Atance和O’Neill, 2001）。

(a) Quintessential example where the target route is found perfectly by the CTM.



(b)一种涌现行为的例子，其中CTM在其训练步数之外沿路径进行关注。

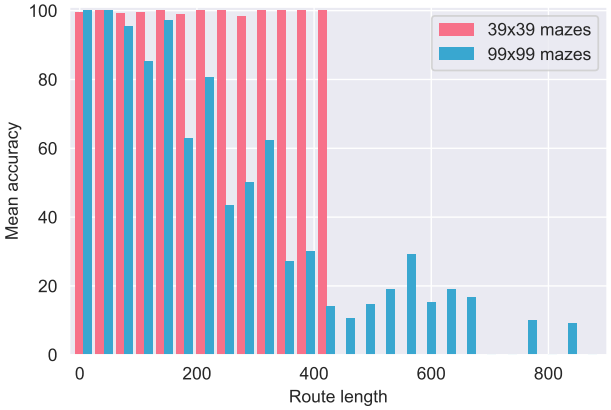
图 8 | 一种自然的迷宫解决方法。每一行展示了 CTM 如何解决一个不同的 39 × 39 迷宫。最左边的图片用彩色箭头展示了注意力的质心在整个思考过程中的变化，展示了 CTM 如何沿着解题路径进行关注。右边的图片是 CTM 输出的解题快照，叠加了注意力热图（颜色与箭头匹配）。(a) 展示了一个典型的例子，而 (b) 则展示了 CTM 即使在训练所用的内部时钟之外，仍然继续沿着路径进行关注（为此，我们让 CTM 比训练时长扩展了 2× 倍）。我们提供了更多的示例，并在项目页面上提供了一个交互演示，您可以在其中与 CTM 互动以解决类似的迷宫。

This CTM 被训练来预测从起始位置向外最多100步。在图8a中我们可以看到，CTM 将注意力集中在迷宫的末端（最右侧的绿色热图；大约75个内部时间片），因为这是它能够预测的范围之内。相比之下，图8b中的真实路径远超过100步。当我们使用比训练时更多的内部时间片来评估注意力图时，注意力模式会继续追踪路径的其余部分：为了生成这些可视化，我们让CTM 运行了2×它被训练的内部时间片。

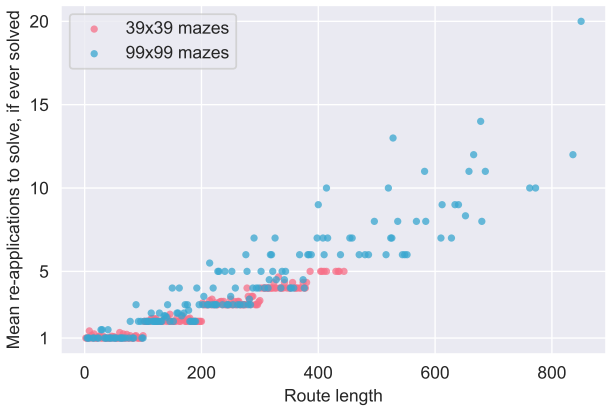
T他的行为是 emergent 并表明 CTM 已学习了一种通用的处理方法
u在基础迷宫任务中，与仅仅 memorizing 训练数据相比。

我在接下来的部分中，我们演示这种CTM如何能够应用于比其训练规模更大的迷宫。
w作为训练集。

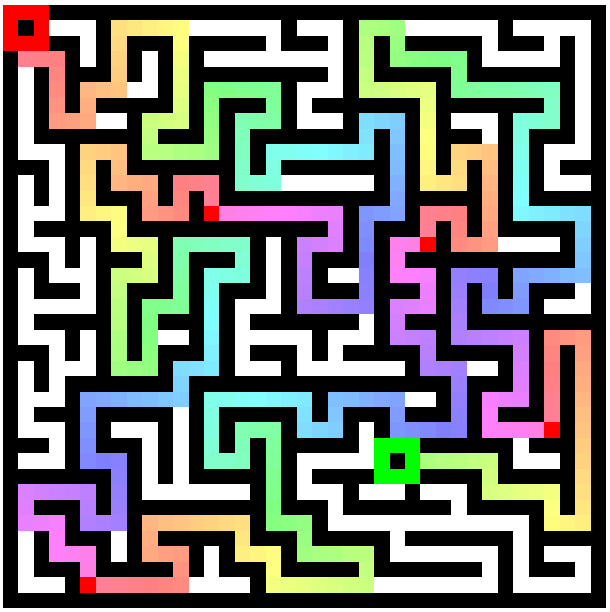
4.3. Generalizing to longer paths and bigger mazes



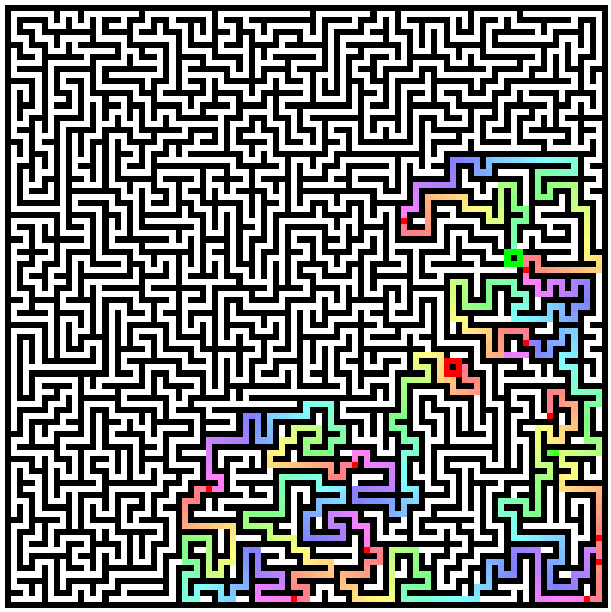
(a) Accuracies when generalizing



(b) Mean re-application count.



(c) Generalizing, 39×39



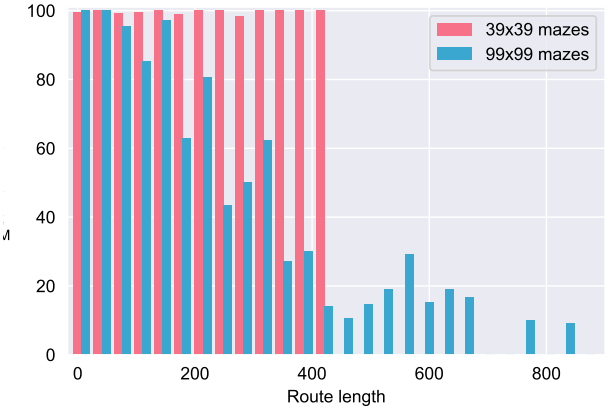
(d) Generalizing, 99×99

Figure 9 | CTM accuracy when generalizing to longer paths and bigger mazes. This CTM was trained to solve mazes of size 39×39, up to a route length of 100 (truncated for longer routes in the training data). In (b) 're-application' occurs when we move the start point to the end of where the CTM predicts for a given maze. We show the generalization examples for (c) 39×39 and (d) 99×99 mazes, where the rainbow of colors (from red to blue) denote the predicted steps per re-application. The initial start and end points are made larger for viewing clarity.

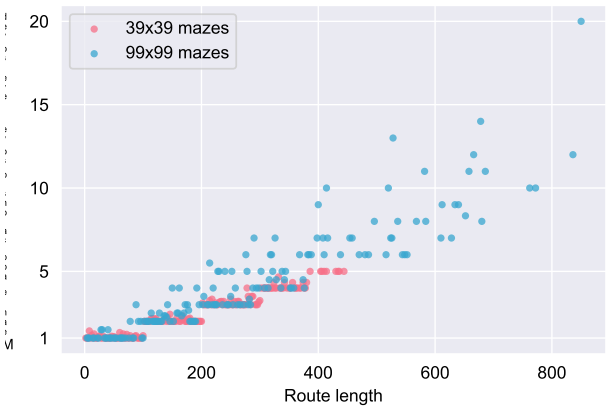
Our observations in the previous section suggested that the CTM might generalize beyond the data it was trained on. One of the reasons we decided to forgo any positional embedding was that such a model could be applied to a maze of any size without changes. To test this we applied the CTM to longer paths and bigger mazes. We set this up as follows:

1. To test for **longer paths** we applied the CTM to the same-sized mazes as were used in training (39×39), but re-applied the CTM whenever a maze was encountered that had a path longer than 100 (that which the CTM was trained to output). For re-application we simply moved the starting point (red pixel) to the **final valid position when following the path output** by the CTM at its final internal tick (of 75).
2. When **generalizing to bigger mazes**, we follow the same protocol as longer paths, but test on mazes of size 99×99.

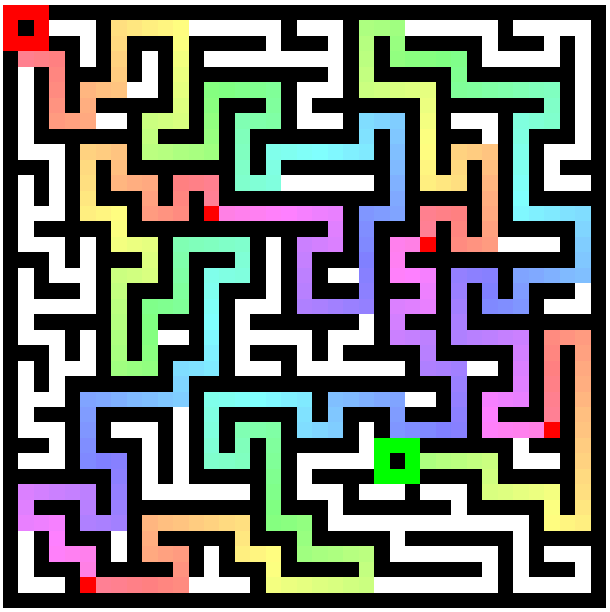
4.3. 将范围推广到更长的路径和更大的迷宫



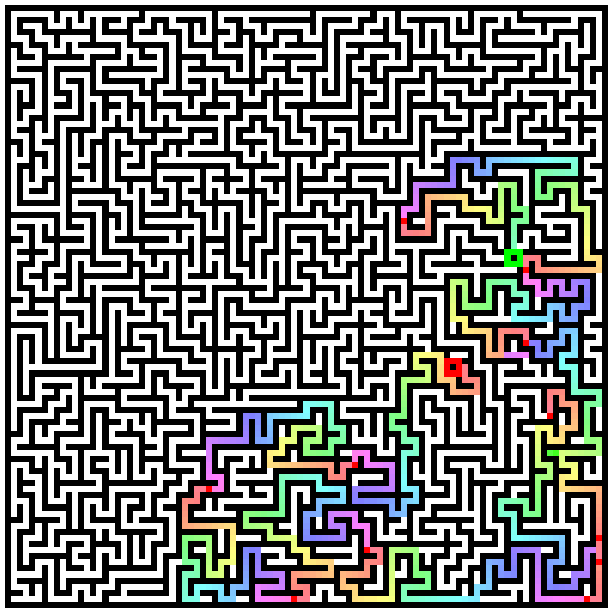
(a) Accuracies when generalizing



(b) Mean re-application count.



(c) Generalizing, 39 × 39



(d) Generalizing, 99 × 99

图 9 | CTM 在泛化到更长路径和更大迷宫时的准确性。该 CTM 被训练解决 39 × 39 大小的迷宫，路径长度最多为 100（训练数据中较长路径被截断）。在 (b) 中，“重新应用”发生在我们将起点移动到 CTM 预测的迷宫末尾。我们展示了 (c) 39 × 39 和 (d) 99 × 99 迷宫的泛化示例，其中颜色渐变（从红色到蓝色）表示每次重新应用的预测步数。初始的起点和终点被放大以便于观察。

我们的观察在上一节中表明，CTM 可能能够泛化到它训练的数据之外。我们决定不使用任何位置嵌入的一个原因是，这样的模型可以应用于任何大小的迷宫而不需要改变。为了测试这一点，我们将 CTM 应用于更长的路径和更大的迷宫。我们设置如下：

1. 为了测试更长的路径，我们将CTM应用到与训练中使用的相同大小的迷宫（39 × 39）上，但每当遇到路径长度超过100（CTM训练输出的长度）的迷宫时，就重新应用CTM。重新应用时，我们只是将起始点（红色像素）移动到按照CTM在其最终内部时钟（75）输出的路径跟随到最后的有效位置。

2. 当推广到更大的迷宫时，我们遵循与较长路径相同的协议，但在大小为 99 × 99 的迷宫上进行测试。

Figure 9 shows the results when generalizing to longer routes and bigger mazes. The CTM performs nearly perfectly for any length route on the 39×39 mazes, but performance begins to taper off for the larger 99×99 mazes. This is probably owing to the larger absolute distances between start and end points for larger mazes. For future work, we expect to explore a continuous training regime that: (1) accounts for the end point predicted by the CTM, (2) keeps the current neural dynamics, (3) 'teleports' the starting point to the predicted end, and (4) continues from there for the next minibatch. Such a setup would be better suited to the sequential nature of the CTM. We encourage readers to use our **interactive demonstration where you can interact with the CTM** to solve mazes like this on our project page. See the future work discussion in Section 12 for a discussion on 'open-world' training.

4.4. Discussion: the need for a world model and cognitive map

Internal models of the world and cognitive maps represent crucial aspects of intelligent systems (Gornet and Thomson, 2024; Ha and Schmidhuber, 2018; LeCun, 2022). In this case, we consider a world model to be an internal representation of the external environment, encapsulating an agent's knowledge about the world's structure, its dynamics, and its actionable place therein. A good world model should enable an agent to reason about the world, plan, and predict the consequence of its actions. Cognitive maps (Gornet and Thomson, 2024) specifically focus on spatial relationships and navigation. The ability to construct and utilize these internal representations is a strong indicator, and arguably a prerequisite, for sophisticated intelligence. The notion of 'episodic future thinking' (Atance and O'Neill, 2001) is even considered a hallmark feature of human intelligence. An agent devoid of a world model would be limited to reactive behaviors. Similarly, lacking a cognitive map would severely restrict an agent's ability to navigate and interact effectively within complex spatial environments. Therefore, the presence and sophistication of world models and cognitive maps can serve as a benchmark for evaluating intelligence.

To this end, we designed the maze task such that it would require a good internal world model to solve. This was achieved by (1) requiring the model to output a route directly, as opposed to solving the maze with a local algorithm (Schwarzschild et al., 2021), and (2) forgoing any positional embedding in the image representation, meaning that the model must build its own spatial cognitive map in order to solve the task (Gornet and Thomson, 2024). Indeed, we saw that the NLMs and synchronization components of the CTM enables it to solve our 2D maze task, far surpassing the best baselines we trained. These results suggest that the CTM is more capable of building and utilizing an internal model of its environment.

5. CIFAR-10: the CTM versus humans and baselines

In this section we test the CTM using CIFAR-10, comparing it to human performance, a feed-forward (FF) baseline, and an LSTM baseline. For the model-based baselines, we used a constrained featurization backbone in order to emphasize the differences owing to the model structure post-featurization (i.e., CTM versus LSTM versus FF). We also used 50 internal ticks to give the CTM and LSTM 'time to think'. We give full architecture details in Appendix E. The human and model baselines were set up as follows:

- **Human baseline**. We used two datasets of human labels for CIFAR-10; we call these CIFAR-10D (Ho-Phuoc, 2018) owing to its calibration of difficulty levels, and CIFAR-10H (Peterson et al., 2019) originally used to quantify human uncertainty.[7] We used CIFAR-10D to determine easy versus difficult samples, and CIFAR-10H as a direct human baseline.

[7] CIFAR-10D can be found at https://sites.google.com/site/hophuoctien/projects/virec/cifar10-classification; CIFAR-10H can be found at https://github.com/jcpeterson/cifar-10h

图9显示了将路径长度延长和迷宫规模扩大后的结果。CTM在39×39迷宫中的任何长度路径上几乎完美地表现，但随着更大规模的99×99迷宫，性能开始逐渐下降。这可能是因为更大规模迷宫中起点和终点之间的绝对距离更大。对于未来的工作，我们预计会探索一种连续训练制度，该制度包括：(1) 考虑CTM预测的终点，(2) 保持当前的神经动力学，(3) 将起始点“传送到”预测的终点，(4) 然后从那里开始下一个小批量。这种设置更适合CTM的序列性质。我们鼓励读者使用我们的交互演示，在那里您可以与CTM互动，解决类似这样的迷宫问题，详见我们项目页面。有关“开放世界”训练的讨论，请参阅第12节的未来工作讨论。

4.4. 讨论：世界模型和认知地图的需求

内部世界模型和认知地图是智能系统的关键方面（Gornet 和 Thomson, 2024; Ha 和 Schmidhuber, 2018; LeCun, 2022）。在这种情况下，我们考虑世界模型是智能体对外部环境的内部表示，包含智能体对世界结构、动力学及其可操作位置的知识。一个好的世界模型应该使智能体能够对世界进行推理、规划，并预测其行为的后果。认知地图（Gornet 和 Thomson, 2024）特别关注空间关系和导航。构建和利用这些内部表示的能力是复杂智能的一个强烈指标，可以说是其先决条件。‘事件性未来思考’的概念（Atance 和 O’Neill, 2001）甚至被认为是人类智能的一个标志性特征。缺乏世界模型的智能体仅限于反应性行为。同样，缺乏认知地图会严重限制智能体在复杂空间环境中的导航和互动能力。因此，世界模型和认知地图的存在及其复杂性可以作为评估智能的标准。

为了实现这一目标，我们设计了迷宫任务，要求解决该任务需要一个良好的内部世界模型。这通过以下方式实现：(1) 要求模型直接输出一条路径，而不是使用局部算法（Schwarzschild et al., 2021）来解决迷宫；(2) 在图像表示中省略任何位置嵌入，这意味着模型必须构建自己的空间认知地图才能解决该任务（Gornet and Thomson, 2024）。确实，我们发现CTM的神经语言模型和同步组件使其能够解决我们的2D迷宫任务，远远超过了我们训练的最佳基线。这些结果表明，CTM更擅长构建和利用其环境的内部模型。

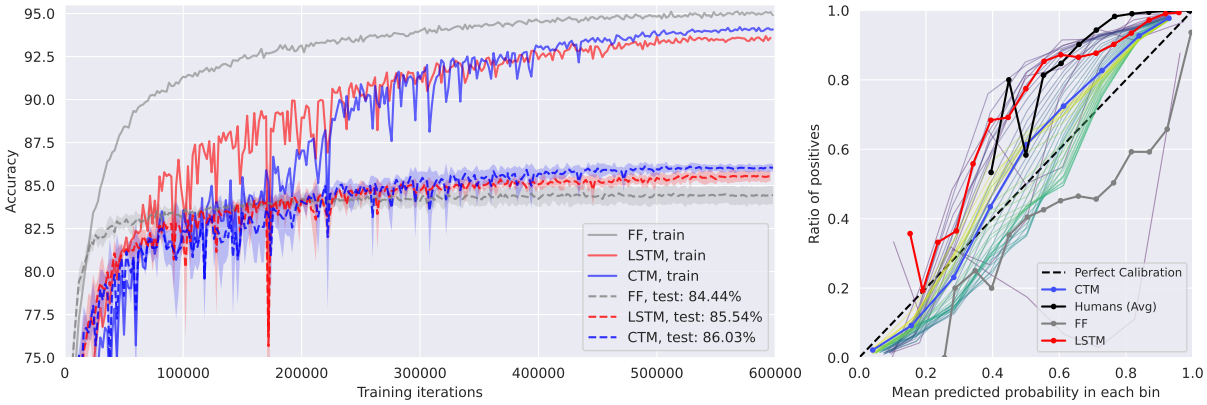
5. CIFAR-10: CTM与人类和基线的对比

在本节中，我们使用CIFAR-10测试CTM，并将其与人类表现、前馈（FF）基线和LSTM基线进行比较。对于基于模型的基线，我们使用了受约束的特征提取骨干，以强调特征提取后的模型结构差异（即CTM与LSTM与FF之间的差异）。我们还使用了50个内部时间片，给CTM和LSTM“思考的时间”。我们在附录E中提供了完整的架构细节。人类和模型基线设置如下：

- Human baseline. 我们使用了两个CIFAR-10的人类标签数据集；我们称这些数据集为CIFAR-10D（Ho-Phuoc, 2018），因为它校准了难度级别，以及CIFAR-10H（Peterson等, 2019），最初用于量化人类不确定性。[7] 我们使用CIFAR-10D来确定容易与困难的样本，而使用CIFAR-10H作为直接的人类基线。

[7] CIFAR-10D can be found at `https://sites.google.com/site/hophuoctien/projects/virec/cifar10-classification`; CIFAR-10H can be found at `https://github.com/jcpeterson/cifar-10h`

- **FF baseline**. A feed-forward only baseline (denoted FF). An MLP was applied to ResNet features after average pooling, where the width of the hidden layer was set to match the parameter count of the CTM for this experiment.
- **LSTM baseline**. An LSTM set up to unroll with an internal thought dimension, with a hidden width set to match the parameter count of the CTM. The LSTM could attend to the image at each step and used the same loss as the CTM for valid comparison.



(a) Training curves. **(b)** Calibration plot.

Figure 10 | CIFAR-10 training curves (average over 3 seeds) and calibration plots for the CTM, a feed-forward only baseline, and an LSTM baseline. The CTM is slower than the LSTM per forward pass (±2.4×) but is also more stable during learning. The CTM has the best test performance. The calibration plot shows that even a human baseline (Peterson et al., 2019) is poorly calibrated, and that the CTM demonstrates good calibration, failing in a way that is strikingly similar to humans.

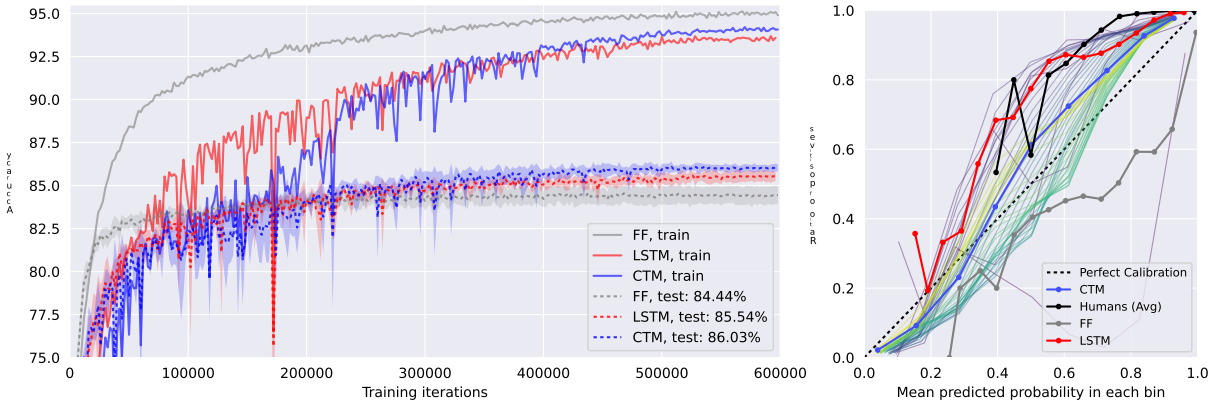
Figure 10 shows the training curves of the CTM, FF, and LSTM models, and calibration plots for each, including an estimation of human calibration using CIFAR-10H. The FF baseline reaches a high training accuracy early on, but also demonstrates a poor generalization gap. The LSTM is less stable during training (we had to set the learning rate to 0.0001 for all experiments because of this) and yields a marginally improved test accuracy. The CTM is more stable and performant.

For the human calibration we used the probabilities provided in CIFAR-10H, which were computed using guesses from multiple humans. We computed calibration here as we did for ImageNet-1K (see Figure 3c): we compute the predictive probability as the average probability for the chosen class over all internal ticks. None of the models are perfectly calibrated, but the CTM demonstrates the best calibration, even when compared to humans. Strikingly, the CTM has even better calibration than humans, while the LSTM follows the human under-confidence.

Figure 11a compares models and CIFAR-10H against the difficulty determined using the CIFAR-10D dataset. Each model and humans have similar trends in this case, although the CTM follows most closely to CIFAR-10H. Figures 11b and 11c compare the uncertainties of the CTM and LSTM to the uncertainties of humans (using reaction times from CIFAR-10H as a proxy for uncertainty). We compute the CTM and LSTM uncertainties using the normalized entropies (see Section 2.5) averaged over internal ticks as this approximates the total uncertainty each model has regarding the observed data. Both the CTM and LSTM exhibit trends similar to human reaction times.

Figure 12 shows the neural activities for the CTM and the LSTM baseline. The CTM yields rich, diverse, and complex dynamics with multiple interesting features, including periodic behavior (there is *no periodic driving function*). The distinct difference between the CTM and LSTM neural activities is evidence that the two novel elements of the CTM (NLMs and synchronization as a representation) enable neural dynamics as a fundamental computational tool.

- FF 基线。一个仅前馈的基线（表示为 FF）。在平均池化之后，应用了一个 MLP 到 ResNet 特征上，其中隐藏层的宽度设置为匹配本次实验中的 CTM 参数数量。
- LSTM 基线。一个配置为展开具有内部思考维度的 LSTM，隐藏宽度设置为与 CTM 的参数数量匹配。LSTM 可以在每一步关注图像，并使用与 CTM 相同的损失函数进行有效的比较。



(a) Training curves. **(b)** Calibration plot.

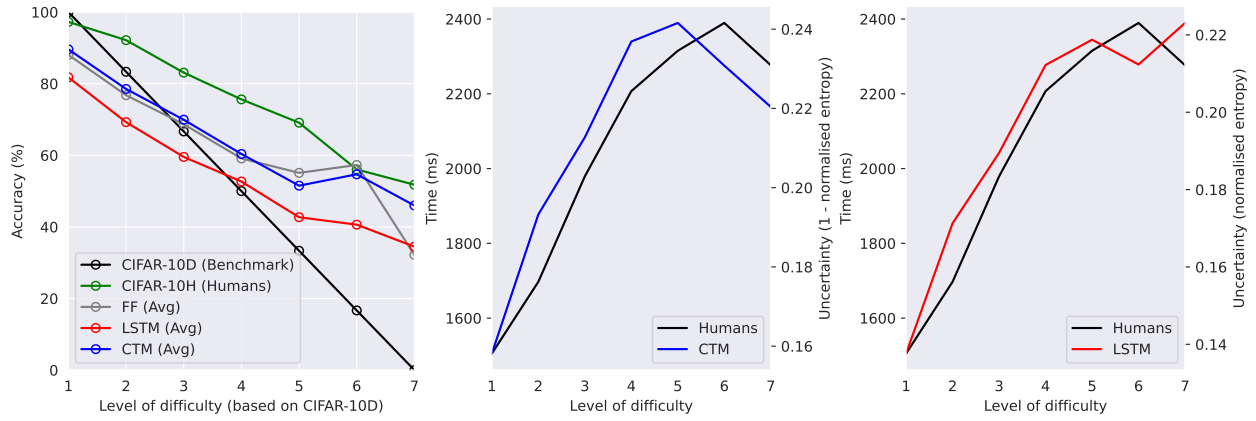
图10 |CIFAR-10训练曲线（3个种子的平均值）和校准图，包括CTM、一个仅前馈基线以及一个LSTM基线。CTM每次前向传播比LSTM慢(±2.4×)，但在学习过程中更加稳定。CTM具有最佳的测试性能。校准图显示，即使是一个人类基线（Peterson et al., 2019）也没有很好地校准，而CTM表现出良好的校准，其失败方式与人类惊人地相似。

图10显示了CTM、FF和LSTM模型的训练曲线，以及每个模型的校准图，包括使用CIFAR-10H进行的人类校准估计。FF基线在训练初期就达到了很高的训练准确率，但也表现出较差的泛化差距。LSTM在训练过程中不够稳定（由于这一点，我们在所有实验中都将学习率设置为0.0001），并获得了略微提高的测试准确率。CTM则更加稳定且性能更佳。

对于人类校准，我们使用了CIFAR-10H提供的概率，这些概率是使用多名人类的猜测计算得出的。我们在这里计算校准的方式与ImageNet-1K相同（参见图3c）：我们计算预测概率为所选类别的内部刻度上的平均概率。没有一个模型是完全校准的，但CTM展示了最佳的校准，即使与人类相比也是如此。令人惊讶的是，CTM的校准甚至比人类更好，而LSTM则表现出人类的欠自信。

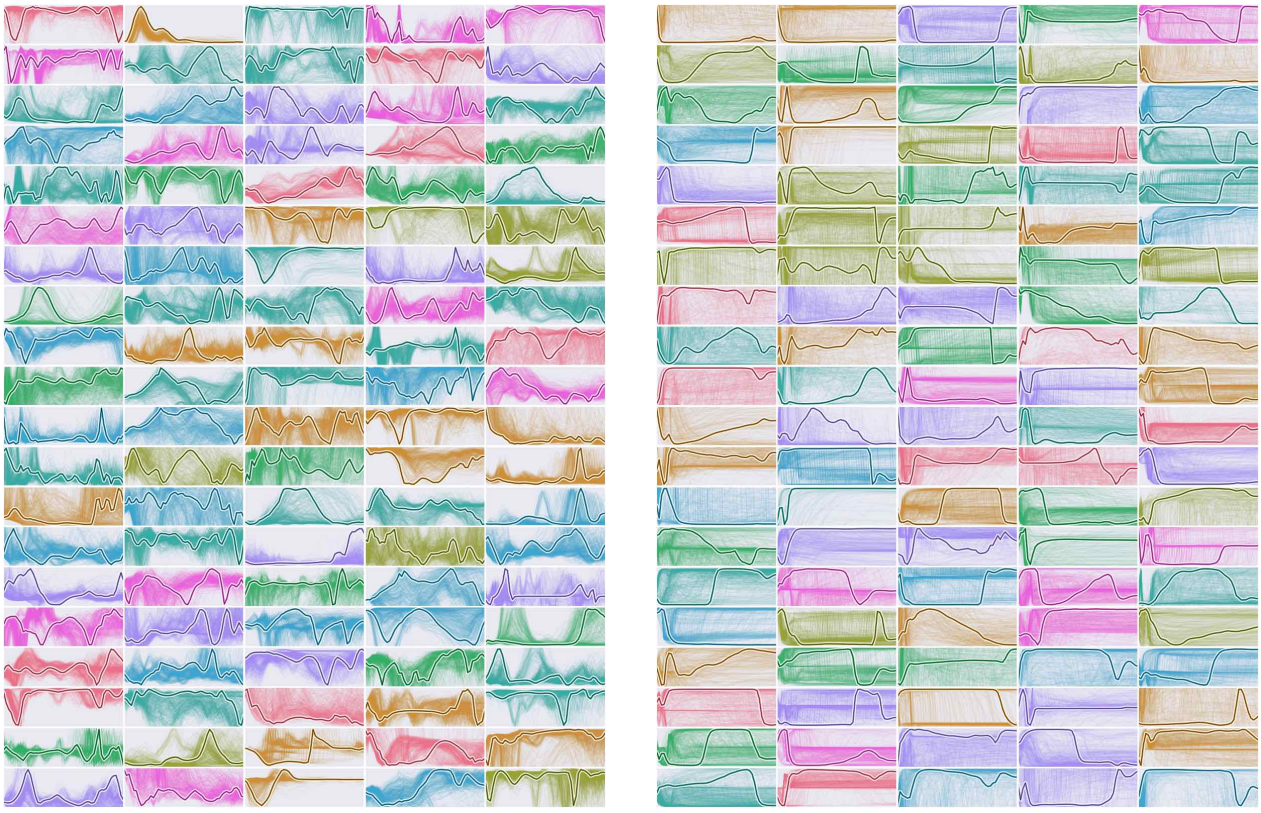
图 11a 将模型和 CIFAR-10H 对比了使用 CIFAR-10D 数据集确定的难度。在这种情况下，每个模型和人类的趋势相似，尽管 CTM 最接近 CIFAR-10H。图 11b 和 11c 将 CTM 和 LSTM 的不确定性与人类的不确定性（使用 CIFAR-10H 的反应时间作为不确定性的代理）进行了比较。我们通过在内部时钟上的归一化熵的平均值（参见第 2.5 节）来计算 CTM 和 LSTM 的不确定性，这大约等于每个模型对所观察数据的总不确定性。CTM 和 LSTM 的趋势与人类的反应时间相似。

图12显示了CTM和LSTM基线的神经活动。CTM产生了丰富、多样且复杂的动力学，包括多种有趣的特征，其中包括周期性行为（存在*no periodic driving function*）。CTM和LSTM神经活动之间的显著差异表明，CTM的两个新颖元素（NLMs和同步作为表示）使神经动力学成为一种基本的计算工具。



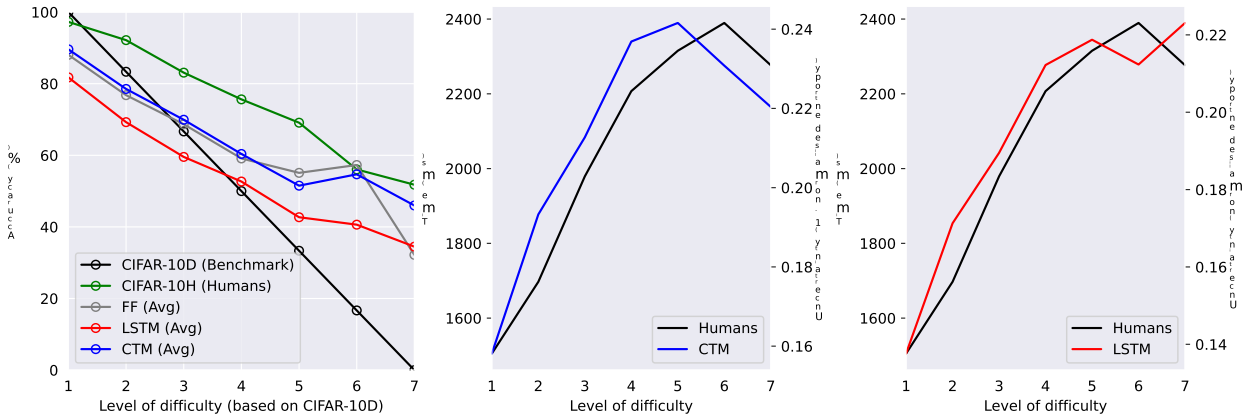
(a) Performance vs. difficulty. **(b)** CTM uncertainty vs. difficulty. **(c)** LSTM uncertainty vs. difficulty.

Figure 11 | Analysis of model and human performance versus difficulty. We used the difficulty calibration from Ho-Phuoc (2018) and compared human predictions from CIFAR-10H (Peterson et al., 2019). We assume that human reaction times are a reasonable proxy for uncertainty and compare this to the trend in uncertainty for the CTM and a parameter-matched LSTM baseline. The error visualized here is a scaled standard deviation.



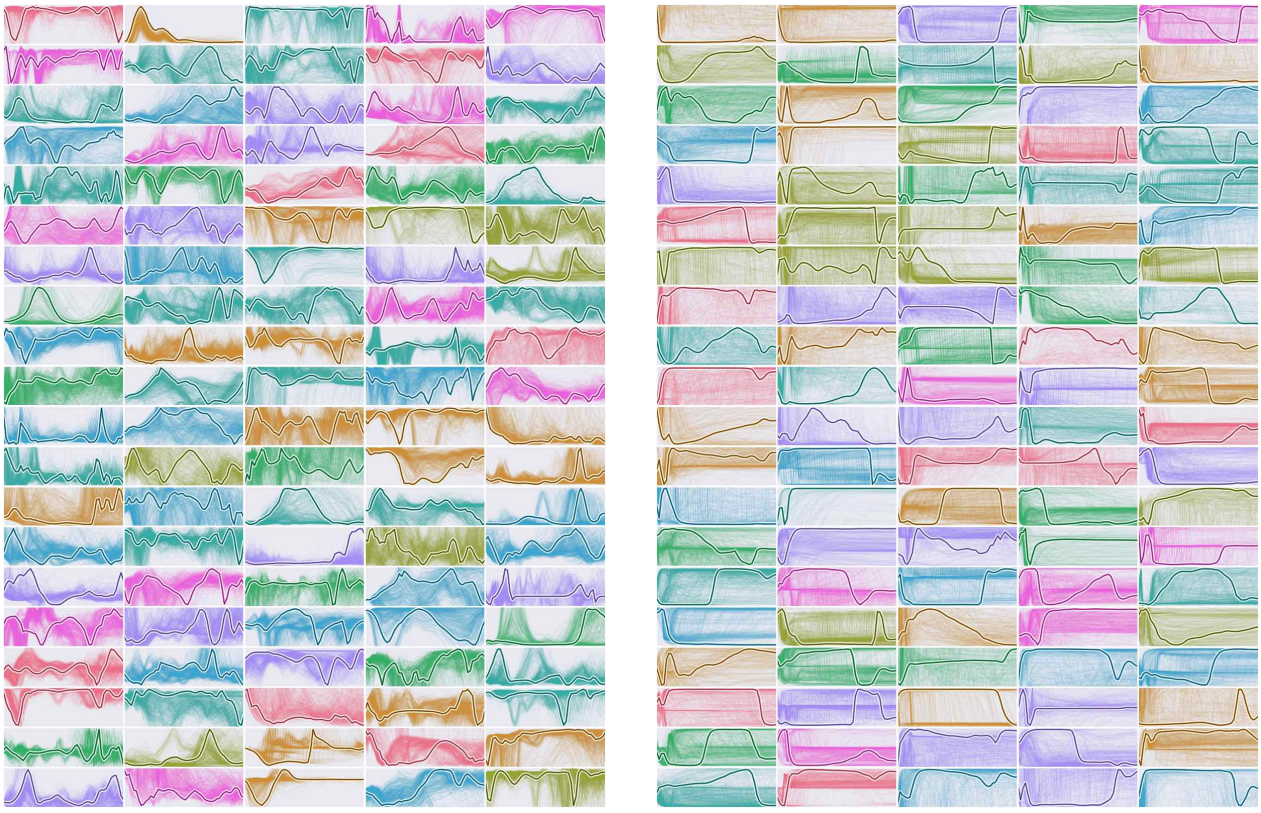
(a) CTM neural activity. **(b)** LSTM neural activity.

Figure 12 | Neuron traces for the CTM and an LSTM baseline, showing how the CTM produces and uses complex neural dynamics even when classifying CIFAR-10. The LSTM yields some dynamic behavior in the post-activation histories shown here, but not nearly to the same degree. Each subplot (in a random color) shows the activity of a single neuron over internal ticks, where multiple examples for different images are shown as faint background lines, and the foreground line is from a randomly chosen example.



(a) Performance vs. difficulty. **(b)** CTM uncertainty vs. difficulty. **(c)** LSTM uncertainty vs. difficulty.

图 11 | 模型和人类表现与难度的分析。我们使用了 Ho-Phuoc (2018) 的难度校准，并将人类预测与 CIFAR-10H (Peterson et al., 2019) 进行了比较。我们假设人类反应时间是不确定性的一个合理代理，并将其与 CTM 和一个参数匹配的 LSTM 基线的不确定性趋势进行比较。这里可视化的是一个放大的标准差。



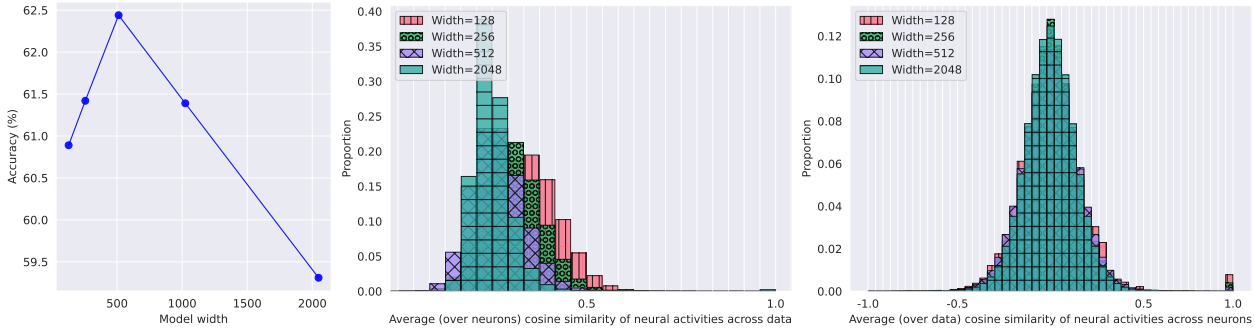
(a) CTM neural activity. **(b)** LSTM neural activity.

图 12 | CTM 和 LSTM 基准的神经元轨迹，展示了即使在分类 CIFAR-10 时，CTM 也产生并使用复杂的神经动力学。LSTM 在这里显示的后激活历史中表现出一些动态行为，但程度远不如 CTM。每个子图（随机颜色）显示了一个神经元在内部计数器上的活动，不同的图像示例作为淡背景线显示，前景线来自随机选择的一个示例。

6. CIFAR-100: ablation analysis

In this section we explore two aspects of the CTM: (1) width (i.e., number of neurons), and (2) number of internal ticks. We used CIFAR-100 in the experiments discussed below as it is a more challenging dataset than CIFAR-10, while remaining relatively low-demand regarding compute.

6.1. Varying the number of neurons



(a) CIFAR-100 accuracies. **(b)** Neuron similarity across data. **(c)** Neuron similarity across neurons.

Figure 13 | CIFAR-100 accuracies and neuron similarities for different model widths. For (b) neuron similarity across data, we computed the average (over neurons) cosine similarities between matched neurons for all pairings across a sample of 128 images – each bar is the proportion of image pairings that have this average neuron similarity. For (c) neuron similarity across neurons, we compute the average (over data) cosine similarities for all pairs of neurons within each model – each bar is the proportion of neurons having that average cosine similarity. Cosine similarity absolutely closer to zero indicates dissimilarity, and hence improved neuron diversity.

Figure 13a shows CIFAR-100 accuracy versus model width (i.e., the number of neurons) for a fixed backbone network (details of which in Appendix F.1), evidencing improved test performance to a point, and then a reduction in performance. The performance drop-off might be related to overfitting, but it might also be that a wider model requires more training (we set a fixed number of training iterations).

Figures 13b and 13c show a relationship between model width and the diversity of neural activity. Intuitively, we expect that with more neurons we would observe a greater degree of neural activity, and these distributions show exactly that. In Figure 13b we see that when measuring cosine similarity on a neuron-level across data points (averaged over all neurons), a wider model results in a tighter distribution around zero. This means that a wider model results in less similar neurons, indicating that the CTM can encode more information about a data point in its neural dynamics when there are more neurons to work with. Figure 13c shows a similar quantity, where we measure the cosine similarity across neurons for the same data points (averaged over many different data points). In this case the wider model only results in a slightly tighter distribution.

6.2. The impact of longer thinking

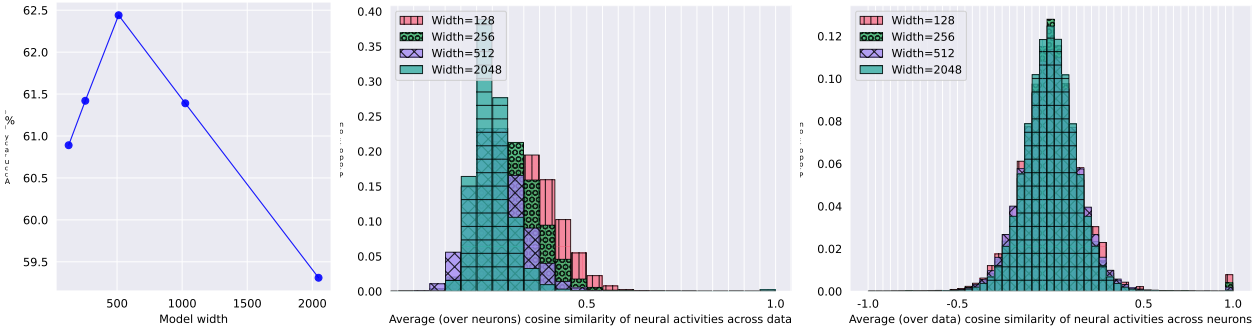
Figure 14 explores the impact of internal ticks on the CTM, showing (a) accuracies versus internal ticks and (b) the distributions over internal ticks where the CTM is most certain. The accuracies in Figure 14a are close, although the CTM using 50 internal ticks was the most performant. This suggests once more that with more internal ticks more training might be warranted.

The emergence of two regions of high certainty in Figure 14b is interesting as it indicates that these CTMs do indeed benefit from having more ‘time to think’, perhaps following two different processes internally depending on the data. Although it is difficult to say exactly why this emerges, the fact that these distributions are far from uniform indicates a more complex process than simply computing the result in a strictly feed-forward nature; more analysis is required in future work.

6. CIFAR-100: 複查分析

我在这一部分我们探讨CTM的两个方面：(1) 宽度（即神经元的数量），和 (2)
n内部时钟的数量。我们在下面的实验中使用了CIFAR-100，因为它更具
c比 CIFAR-10 更具有挑战性的数据集，同时在计算需求上保持相对较低。

6.1. 神经元数量的变化



(a) CIFAR-100 accuracies. **(b)** Neuron similarity across data. **(c)** Neuron similarity across neurons.

图 13 | 不同模型宽度下的 CIFAR-100 准确率和神经元相似度。对于 (b) 数据间的神经元相似度，我们计算了样本中 128 张图像的所有配对匹配神经元之间的余弦相似度的平均值（按神经元计算）——每根柱状图表示具有这种平均神经元相似度的图像配对的比例。对于 (c) 神经元间的神经元相似度，我们计算了每个模型内所有神经元配对的余弦相似度的平均值（按数据计算）——每根柱状图表示具有这种平均余弦相似度的神经元的比例。余弦相似度绝对值越接近零表示越不相似，因此提高了神经元的多样性。

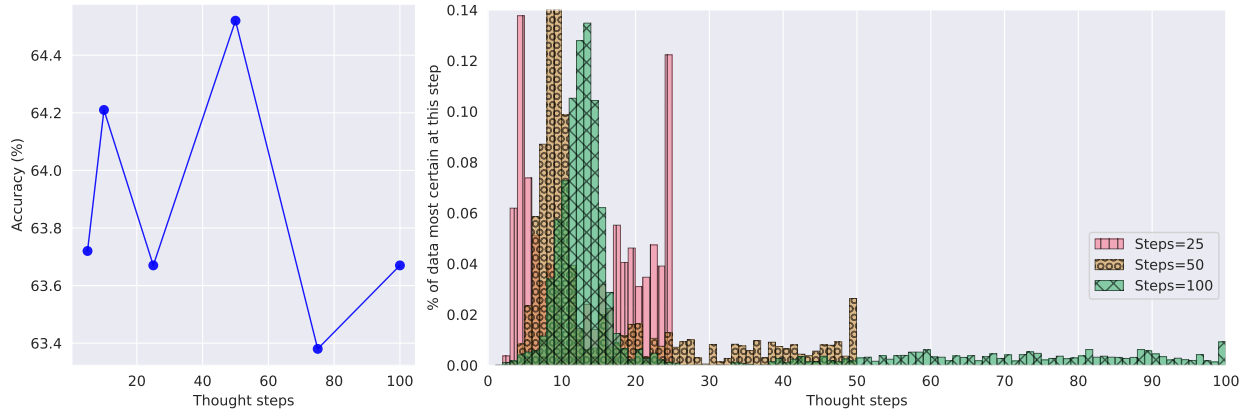
图13a展示了固定主网络（详细内容见附录F.1）的CIFAR-100准确率与模型宽度（即神经元的数量）之间的关系，显示出性能在某个点上有所提升，之后则出现下降。性能下降可能与过拟合有关，但也可能是更宽的模型需要更多的训练（我们设定了固定的训练迭代次数）。

图13b和图13c展示了模型宽度与神经活动多样性的关系。直观地讲，我们期望随着神经元数量的增加，会观察到更广泛的神经活动，并且这些分布确实显示了这一点。在图13b中，我们看到在数据点上（所有神经元的平均值）测量神经元级别的余弦相似度时，更宽的模型会导致零附近的分布更紧密。这意味着更宽的模型会导致神经元之间的相似度降低，表明CTM在有更多的神经元可用时，能够通过其神经动力学编码更多关于数据点的信息。图13c显示了类似的数量，我们测量了相同数据点（许多不同数据点的平均值）上神经元之间的余弦相似度。在这种情况下，更宽的模型仅导致分布稍微更紧密。

6.2. 更长思考的影响

图14探讨了内部时钟对CTM的影响，显示了(a)准确率与内部时钟的关系以及(b)CTM最确定的内部时钟分布情况。图14a中的准确率相近，尽管使用50个内部时钟的CTM表现最佳。这再次表明，随着内部时钟数量的增加，可能需要更多的训练。

图14b中出现两个高确定性区域是有趣的，因为它表明这些CTMs确实可以从有更多“思考时间”中受益，也许内部依赖于不同的过程，这取决于数据。虽然很难确切地说出这是为什么，但这些分布远非均匀分布的事实表明，这不仅仅是以严格前馈的方式计算结果；未来的工作需要进行更多的分析。



(a) CIFAR-100 accuracies.

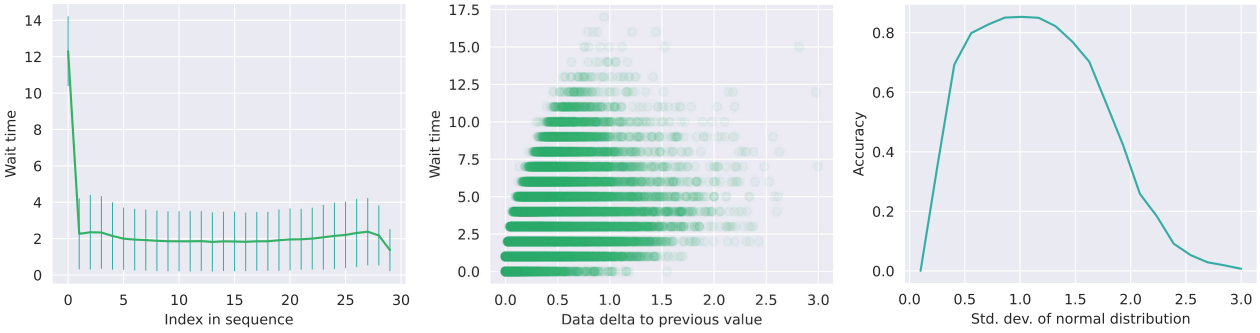
(b) Distribution of where the CTM is most certain.

Figure 14 | CIFAR-100 accuracies and internal tick analysis. The distributions and accuracies in (b) are computed for those internal ticks (x-axis) where the CTM's were the most certain (see Section 2.5). In each case the CTM has two regions of certainty, early on and later, regardless of how many internal ticks are used.

7. Sorting

In this section, we apply the CTM to the task of sorting 30 numbers drawn from the normal distribution. Sorting real numbers was a task explored by Graves (2016) when designing RNNs for adaptive compute, and it provides a test bed for understanding the role of compute for an adaptive-compute system, such as the CTM. In this case the CTM does not use attention, but rather ingests the randomly shuffled input data (30 real numbers) directly. This is implemented by replacing the attention mechanism with a straightforward concatenation, replacing (10) in Figure 1.

Can the CTM produce sequential outputs? For this experiment we set the CTM up to output a sequence over its internal ticks. This is a more standard approach to modeling sequences and we wanted to understand whether the CTM could be trained in this fashion. At each internal tick the CTM output a vector of length 31, including 30 indices for sorting and the 'blank' token used for the well-known connectionist temporal classification (CTC) loss (Graves et al., 2006). We then applied this CTC loss over the full output of the CTM over its internal ticks.

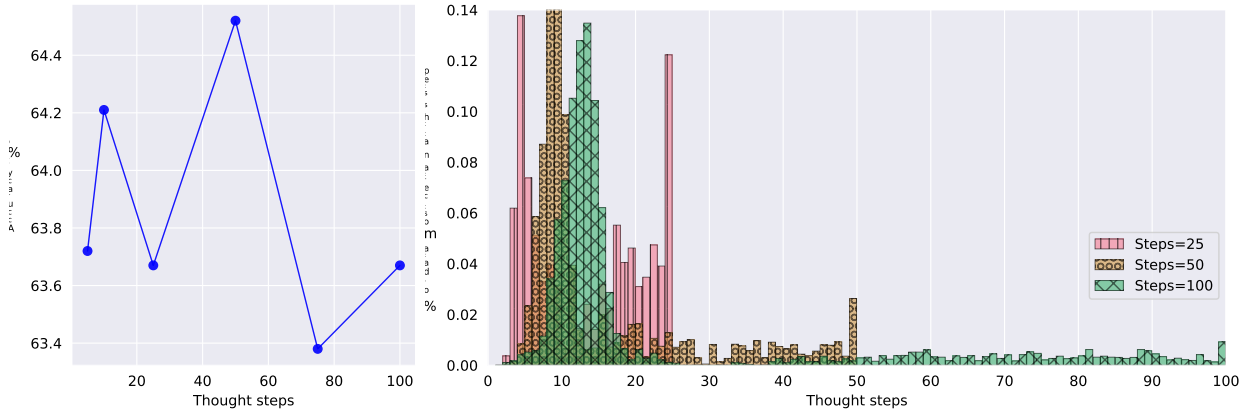


(a) Mean wait times per sequence index (measured in ticks).

(b) Wait times versus change in sequence values.

(c) Generalizing beyond the distribution used for training.

Figure 15 | Results when sorting on $\mathcal{N}(0, I_{30})$. In (a) we can see an evident pattern in the average wait times, where the initial wait time (number of internal ticks) is high, goes to its lowest point, and has a slightly higher bump toward the end of the sequence. In (b) we see that the CTM employs various wait times, but that the difference between the previous output value and the current output value ('data delta') impacts wait time. In (c) we see how this CTM can scale to data drawn from different normal distributions.



(a) CIFAR-100 accuracies.

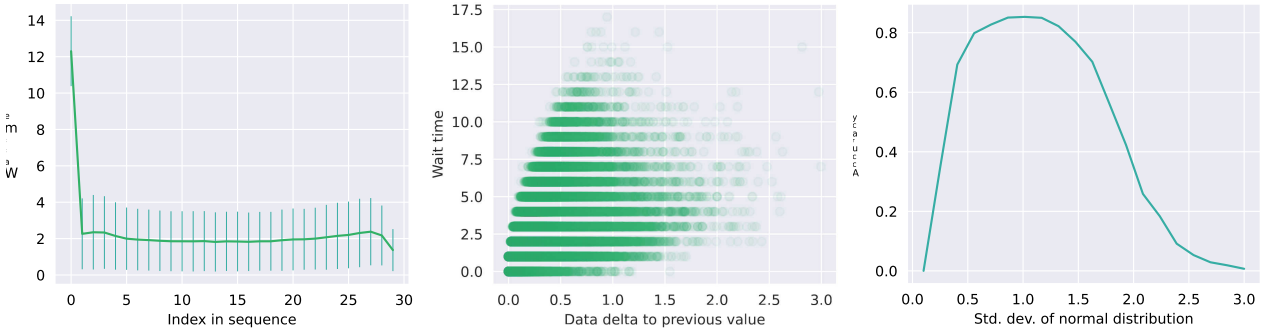
(b) Distribution of where the CTM is most certain.

图 14 | CIFAR-100 准确率和内部刻度分析。在 (b) 中，这些分布和准确率是在 CTM 最为确定的那些内部刻度（x 轴）处计算的（参见第 2.5 节）。无论使用多少内部刻度，CTM 始终有两个确定性区域，早期和后期。

7. 排序

在本节中，我们将CTM应用于从正态分布中抽取的30个数字的排序任务。排序实数是Graves（2016年）在设计用于自适应计算的RNN时探索的任务之一，它为理解自适应计算系统（如CTM）的计算角色提供了一个测试床。在这种情况下，CTM不使用注意力机制，而是直接摄入随机打乱的输入数据（30个实数）。这通过用简单的串联操作替换注意力机制在图1中实现。

CTM能否生成序列输出？对于这个实验，我们将CTM设置为在其内部时钟上输出一个序列。这是一种更标准的序列建模方法，我们想了解CTM是否可以以这种方式进行训练。在每个内部时钟点，CTM输出一个长度为31的向量，包括30个用于排序的索引和用于连接主义时序分类（CTC）损失的“空白”标记（Graves等，2006）。然后，我们在CTM的整个内部时钟输出上应用了这种CTC损失。



(a) Mean wait times per sequence index (measured in ticks).

(b) Wait times versus change in sequence values.

(c) Generalizing beyond the distribution used for training.

图 15 | 按 $\mathcal{N}(0, I_{30})$ 排序时的结果。在 (a) 中，我们可以看到平均等待时间中有一个明显的模式，初始等待时间（内部时钟数）较高，然后降到最低点，并在序列末尾有一个轻微的上升。在 (b) 中，我们看到 CTM 使用了各种等待时间，但前一个输出值与当前输出值之间的差异（“数据变化量”）影响了等待时间。在 (c) 中，我们看到这种 CTM 可以适应来自不同正态分布的数据。

Figure 15 gives the results of the CTM on the sorting task. There is a clear pattern to the process it follows, as evidenced by a correlation between wait times and both the current sequence index (a) and the difference between the previous value and the current value being output (b). A similar task was explored by Graves (2016), who sorted 15 numbers using an adaptive compute RNN. In their case, they observed similar wait times before beginning output (analogous to our first sequence element) and also near the end of the sequence. Our analysis of the relationship between wait times and the difference between current and previous data values (what we call 'data delta' in Figure 15b constitutes evidence that the CTM is using an internal algorithm that depends on the layout of the data. We also show that this CTM generalizes to distributions outside of the training data.

Figure 16 demonstrates the CTM's wait times in a real use-case. The red bars indicate longer than average wait times for a given index, and green bars indicate shorter than average wait times. Longer wait times tend to be related to bigger gaps between data points ('data delta' in Figure 15b).

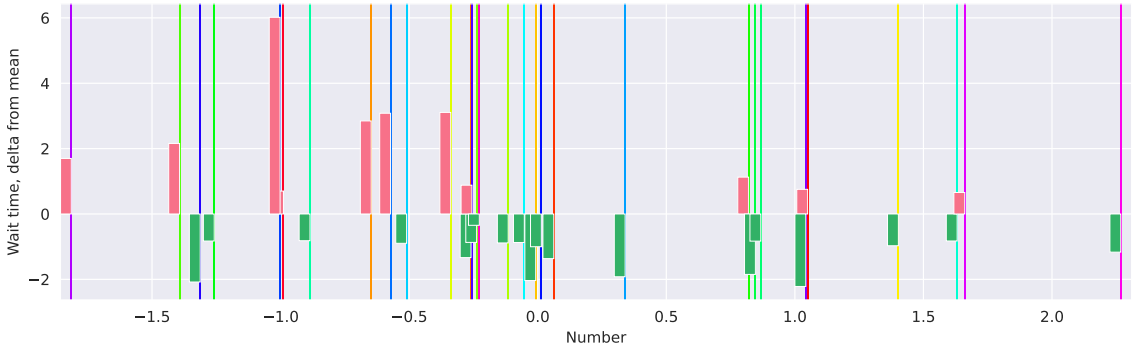
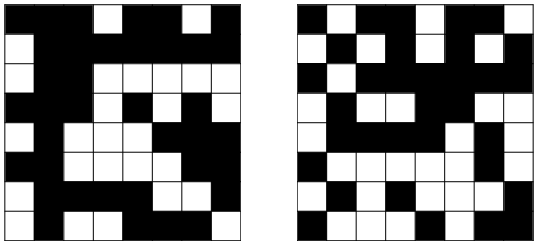


Figure 16 | Sorting demonstration. The input data is represented as vertical lines whose colors denote their original shuffled position (from purple through to red in the 'rainbow' colormap). The red and green bars show positive and negative deviation from the mean wait time (Figure 15a for each index in the sequence), respectively.

8. Parity



(a) Input. **(b)** Target.

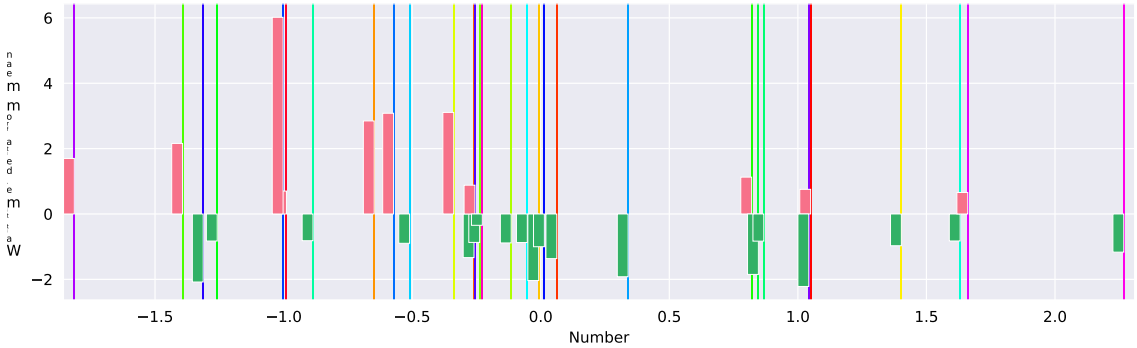
Figure 17 | Parity task. The input (a) is a sequence of 64 binary values (top left to bottom right), and the target (b) is the cumulative parity at each position. Here □ indicates positive parity and ■ indicates negative parity.

The parity of a binary sequence is given by the sign of the product of its elements. When processing a sequence element by element, an RNN could conceivably compute parity by maintaining an internal state, flipping an internal 'switch' whenever a negative number is encountered. However, if the entire sequence is provided simultaneously, the task increases in difficulty due to the increasing number of distinct patterns in the input. Previous work (Graves, 2016) has addressed this challenge using recurrent models, which can learn sequential algorithms for statically presented data. Computing parity, as posed in this manner, is well-suited for testing the capabilities of the CTM.

We apply the CTM to the task of computing the parity of a 64-length sequence containing the values 1 and -1 at random positions. Unlike Graves (2016), we set up the task such that the model computes

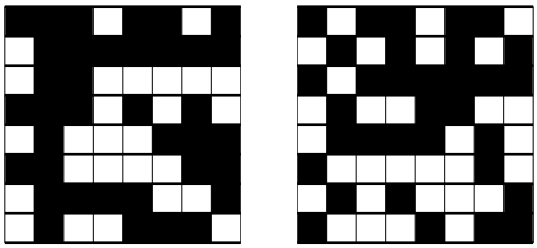
图15给出了CTM在排序任务上的结果。它遵循一个清晰的过程模式，这可以通过等待时间与当前序列索引（a）以及输出的当前值与先前值之间的差异（b）之间的相关性得到证实。Graves（2016）也研究了类似的任务，他们使用自适应计算RNN对15个数字进行了排序。在他们的情况下，他们观察到类似等待时间在开始输出之前（类似于我们的第一个序列元素）以及序列的末尾附近。我们对等待时间与当前值和先前值之间差异的关系的分析（我们在图15b中称之为“数据delta”）表明，CTM正在使用一个依赖于数据布局的内部算法。我们还展示了这种CTM可以泛化到训练数据之外的分布。

图16展示了CTM在实际使用案例中的等待时间。红色条形图表示给定索引比平均等待时间长的等待时间，而绿色条形图表示比平均等待时间短的等待时间。较长的等待时间通常与数据点之间的较大差距（如图15b中的“数据delta”）相关。



F图16 | 排序演示。输入数据用垂直线表示，颜色表示其原始
shuffled位置（从紫色到红色在“彩虹”色图中）。红色和绿色条形图显示正负值。
d从均等待时间偏离（图15a中每个序列索引处），分别。

8. 奇偶性



(a) Input. **(b)** Target.

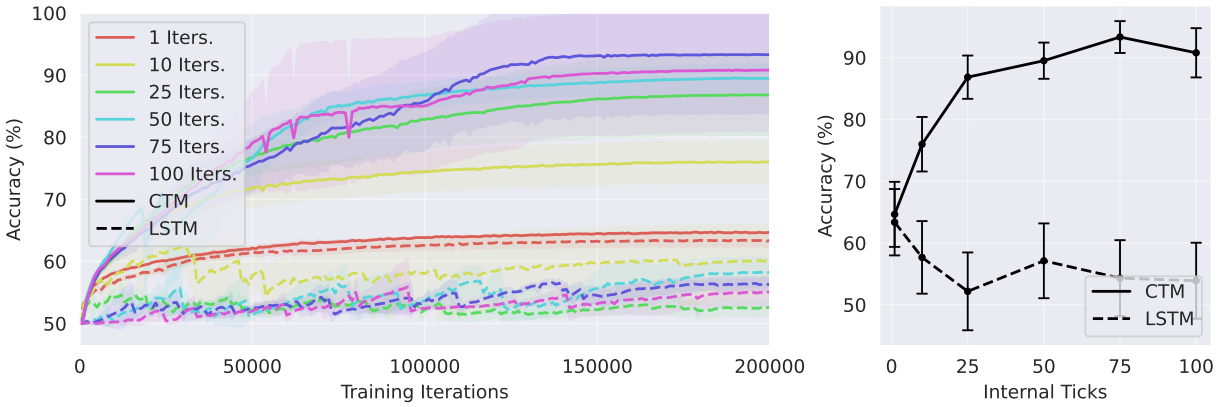
F图17 | 奇偶校验任务。输入(a)是一个由64个二进制值组成的序列（从左上到右下），目标(b)是
c累计奇偶性在每个位置。这里□表示正奇偶性，■表示负奇偶性。

二进制序列的奇偶性由其元素乘积的符号给出。逐元素处理序列时，RNN 可以通过维护内部状态并在遇到负数时翻转内部“开关”来计算奇偶性。然而，如果整个序列同时提供，任务会因输入中不同模式数量的增加而变得更加困难。以往的工作（Graves, 2016）已经使用循环模型解决了这一挑战，这些模型可以学习静态呈现数据的序列算法。以这种方式计算奇偶性，非常适合测试CTM的能力。

W我们应用CTM来计算包含64个值的序列的奇偶校验
1 并在随机位置插入 -1。与Graves (2016) 不同，我们将任务设置为模型计算

the cumulative parity at every index of the sequence, not just the final parity. An example is shown in Figure 17. The values -1 and 1 are embedded as learnable vectors combined with positional embeddings, using attention to ingest input data. We train the CTM with the loss function described in Section 2.5. As a baseline we also trained an LSTM, but set t_2 to be the final iteration since this gave the best results and stability for LSTM training. See Appendix G for more details.

8.1. Results

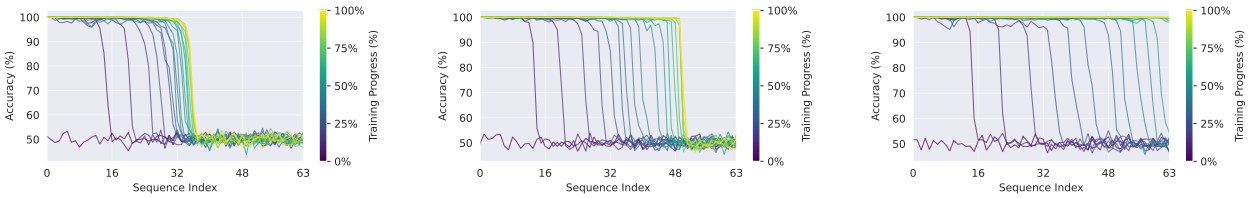


(a) Training curve for CTM and LSTM configurations.

(b) Accuracy vs internal ticks.

Figure 18 | The training curve (left) and the final accuracy vs internal ticks (right) for various CTM and LSTM configurations. The shaded areas and error bars represent one standard deviation across seeds. For the CTM, increased thinking time leads to an increase in performance.

Accuracy increases with thinking time. Figures 18a and 18b show the training curves and final accuracies for various configurations of the CTM, where we changed the number of internal ticks (T) and memory length (M). We also plot parameter-matched LSTM baselines for comparison. Generally, the accuracy of the CTM improves as the number of internal ticks increases. The best-performing models were CTMs with 75 or 100 internal ticks, which could reach 100% accuracy in some seeded runs. The LSTM baselines, on the other hand, struggle to learn the task, with the best performing LSTM, which had 10 internal ticks, achieving an accuracy of 67% ± 0.05%. The LSTM baselines with over 10 internal ticks demonstrate unstable learning behavior; this mimics our observations in Section 4.1, that simple recurrent models are not necessarily well-suited to unfolding an internal thought process. Although the CTM exhibits much more stable training, there is considerable variance in final accuracies due to the choice of random seed. This is discussed in more detail in Appendix G.4.



(a) 10 internal tick CTM.

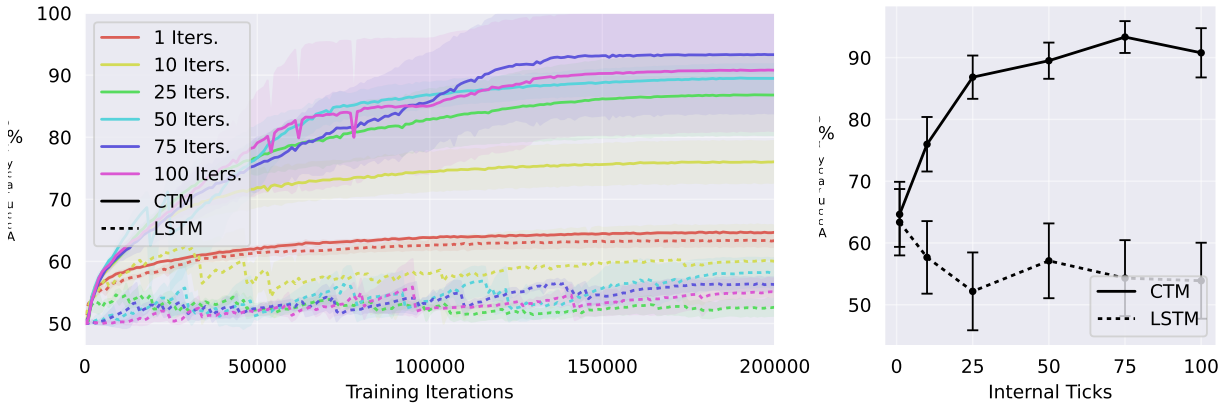
(b) 25 internal tick CTM.

(c) 75 internal tick CTM.

Figure 19 | Accuracy across the 64-element sequence at different training stages (indicated by color) for various internal tick configurations. Early in training, all CTMs accurately predict parity only for initial sequence elements, gradually improving for later elements as training progresses. Models with more internal ticks achieve higher accuracy, with the 10-step model (a) correctly predicting approximately half the sequence and the 75-step model (c) correctly predicting the entire cumulative parity sequence.

在序列的每一个索引处的累积奇偶性，而不仅仅是最终的奇偶性。示例如图17所示。值-1和1被嵌入为可学习向量与位置嵌入结合，并使用注意力机制来摄取输入数据。我们使用第2.5节中描述的损失函数训练CTM。作为基线，我们还训练了一个LSTM，但将t_2设置为最终迭代，因为这在LSTM训练中给出了最佳结果和稳定性。更多细节请参见附录G。

8.1. 结果

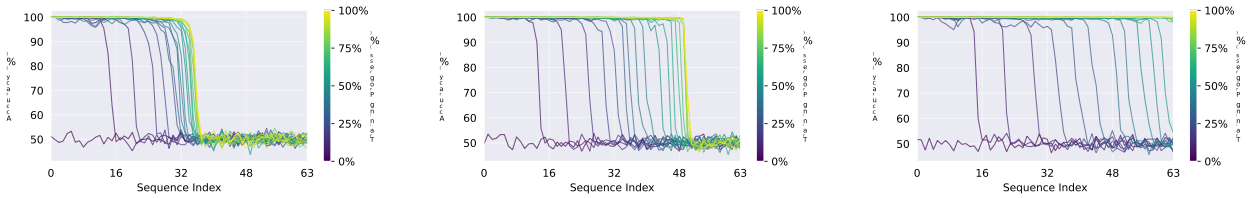


(a) Training curve for CTM and LSTM configurations.

(b) Accuracy vs internal ticks.

F图 18 | 各种CTM和LSTM配置的训练曲线（左）和最终准确率与内部时钟的关系（右）。T他所标注的区域和误差棒表示 Across seeds 的一个标准偏差。对于 CTM，增加思考时间导致to 性能的提高。

准确度随着思考时间的增加而提高。图18a和18b展示了不同CTM配置的训练曲线和最终准确度，其中我们改变了内部时钟数(T)和记忆长度(M)。我们还绘制了参数匹配的LSTM基线进行比较。总体而言，随着内部时钟数的增加，CTM的准确度有所提高。表现最好的模型是具有75或100个内部时钟的CTM，在某些种子运行中可以达到100%的准确度。另一方面，LSTM基线难以学习该任务，表现最好的LSTM，具有10个内部时钟，准确度为67% ± 0.05%。具有超过10个内部时钟的LSTM基线显示出不稳定的训练行为；这与我们在第4.1节中的观察相符，即简单的递归模型不一定适合展开内部思维过程。尽管CTM的训练更为稳定，但由于随机种子的选择，最终准确度存在相当大的差异。这在附录G.4中进行了更详细的讨论。



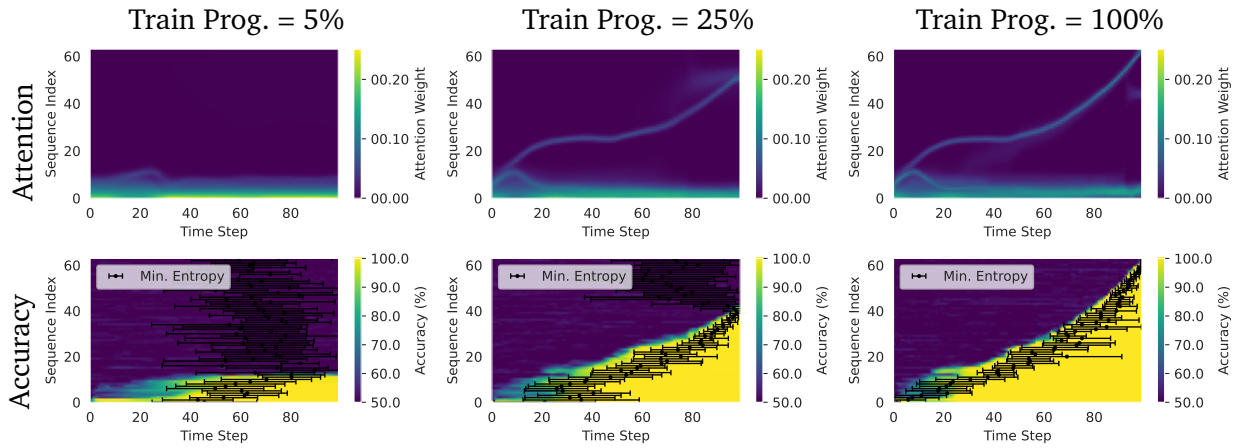
(a) 10 internal tick CTM.

(b) 25 internal tick CTM.

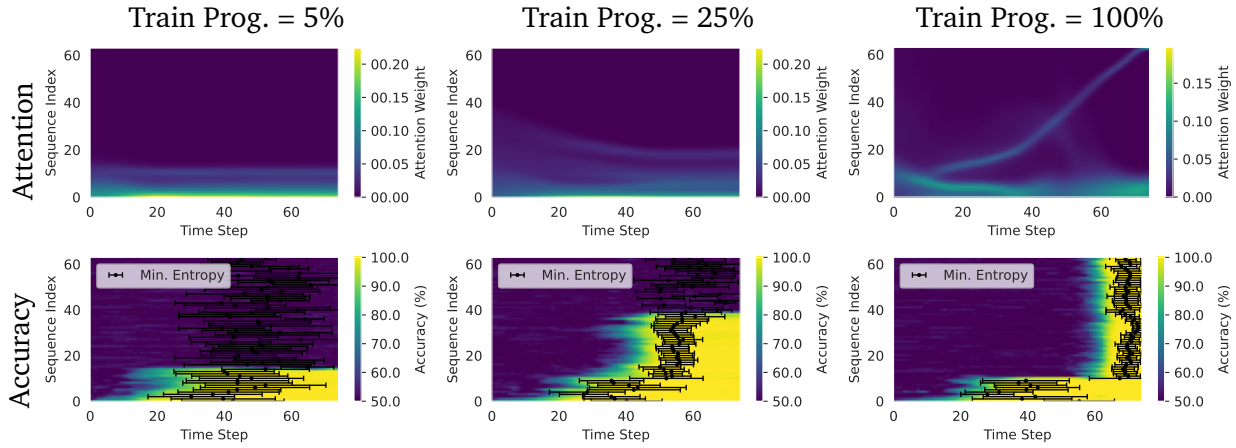
(c) 75 internal tick CTM.

图19 | 在不同训练阶段（用颜色表示）下，64个元素序列的准确率变化情况，以及各种内部计时器配置。在训练初期，所有CTM仅准确预测初始序列元素的奇偶性，随着训练的进行，逐渐提高对后续元素的预测准确性。具有更多内部计时器的模型实现更高的准确率，10步模型（a）正确预测了大约一半的序列，而75步模型（c）正确预测了整个累积奇偶性序列。

The CTM learns a sequential algorithm. To analyze how the CTM learns to solve the parity task, Figure 19 shows the accuracy for each of the 64 elements in the input sequence at different stages of training, for three different internal tick configurations. The models first learn to predict the parity of the initial elements, and as training proceeds, learn to predict later and later positions. With more internal ticks, the model can accurately predict more elements in the target sequence.



(a) CTM trained with 100 internal ticks.

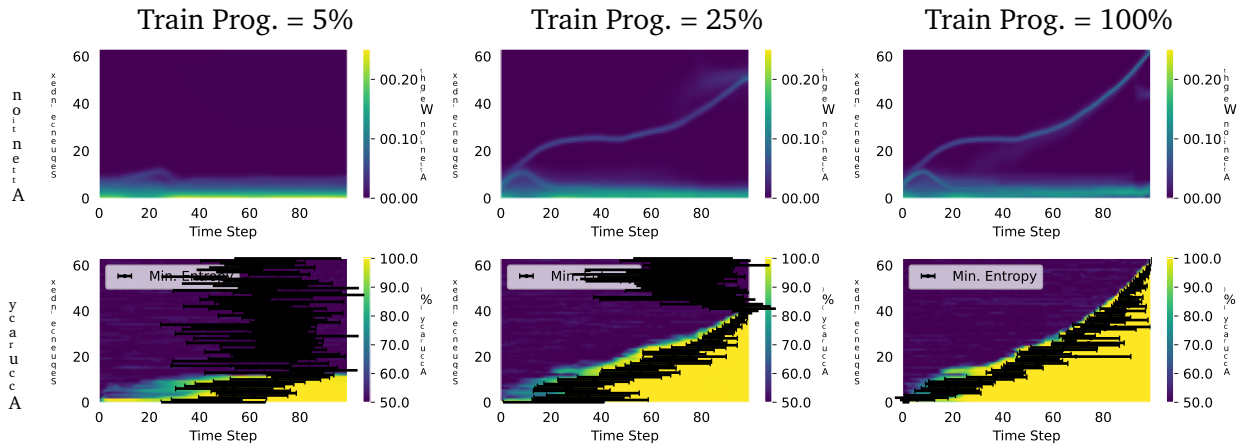


(b) CTM trained with 75 internal ticks.

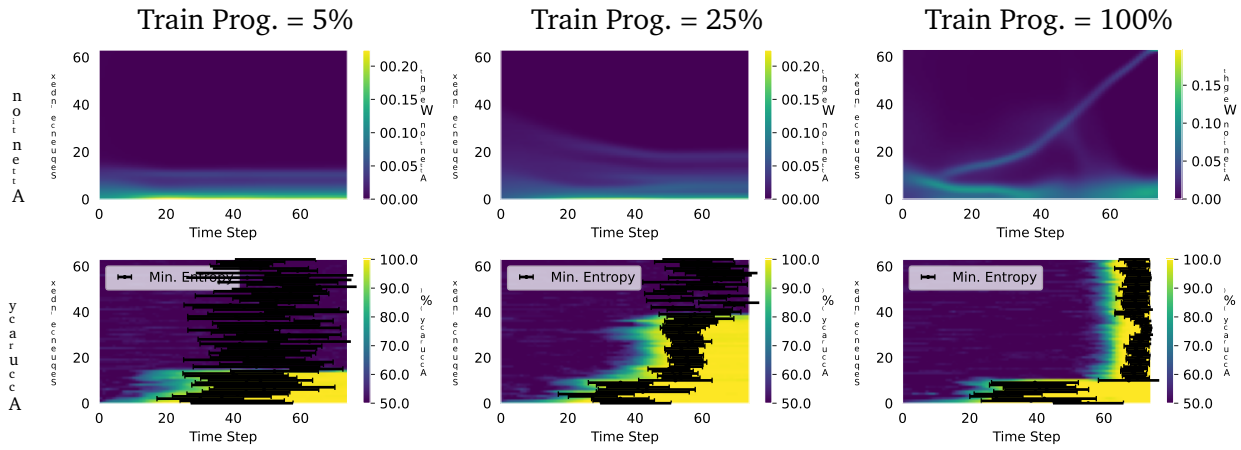
Figure 20 | Attention patterns (top) and accuracy (bottom) at different points in training, for a CTM trained with 100 internal ticks (a) and 75 internal ticks (b). The black points in the accuracy plots denote the internal tick at which the model reached maximum certainty, with the error bars denoting one standard deviation across samples.

To gain insight into how the model solves the cumulative parity task, we visualize the CTM's attention patterns, accuracy, and points of highest certainty across all 64 elements at multiple stages of training in Figure 20 for two different models. The attention and certainty patterns evidence that these CTMs are leveraging different algorithms to solve the cumulative parity task. When using 100 internal ticks, attention moves from the beginning to the end of the sequence, and with it, the model increases its certainty of the prediction at that position. The CTM with 75 iterations, on the other hand, learns to attend to the sequence in reverse order, accurately predicting the parity of the majority of the sequence simultaneously during the final internal ticks. This reverse search through the data suggests that the CTM is carrying out a form of planning, building up its understanding of the observed data before making a final decision on the cumulative parity of the sequence. These results highlight that although multiple strategies exist for solving this task, some of which are more interpretable than others, the CTM clearly demonstrates the ability to form and follow a strategy.

CTM 学习一个序列算法。为了分析 CTM 如何学习解决奇偶性任务，图 19 显示了在不同训练阶段，输入序列中每个元素（共 64 个元素）的准确度，对于三种不同的内部时钟配置。模型首先学会预测初始元素的奇偶性，随着训练的进行，学会预测越来越晚的位置。内部时钟越多，模型可以更准确地预测目标序列中的更多元素。



(a) CTM trained with 100 internal ticks.



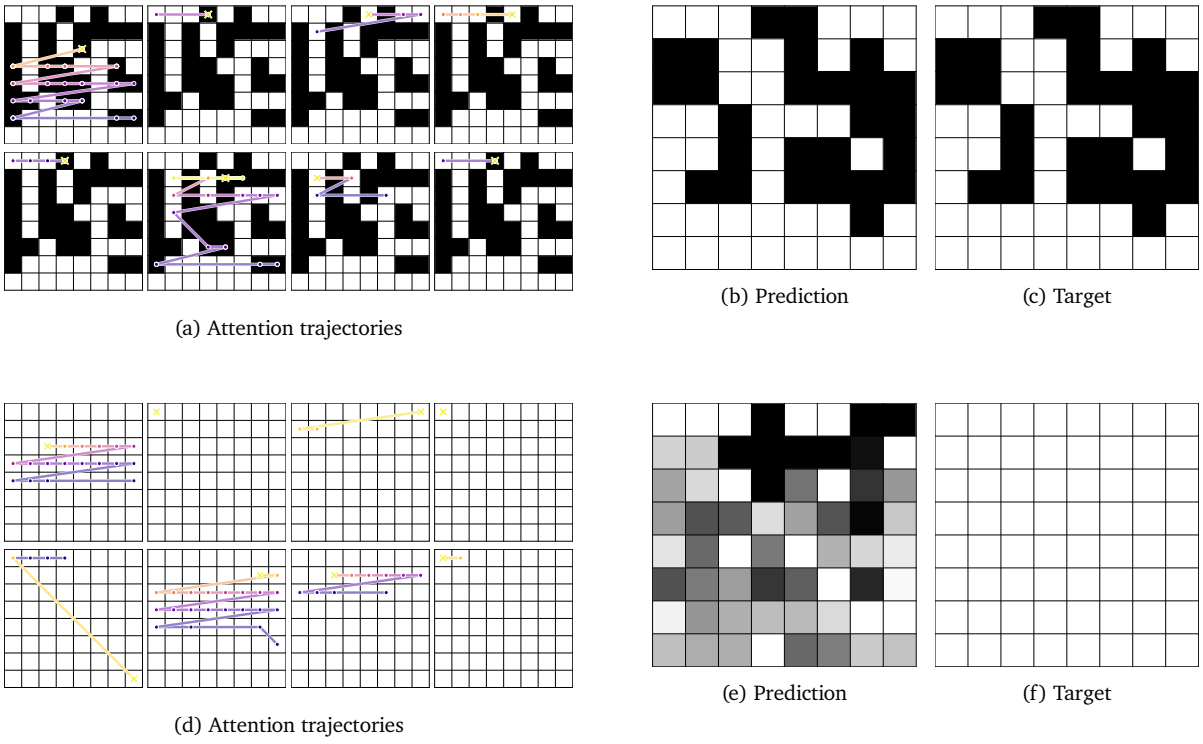
(b) CTM trained with 75 internal ticks.

图20 | 训练不同阶段的注意力模式（顶部）和准确性（底部），对于使用100个内部节拍（a）和75个内部节拍（b）训练的CTM。准确性图表中的黑色点表示模型达到最大确定性的内部节拍，误差棒表示样本间的标准偏差。

为了深入了解模型如何解决累积对等任务，我们在图20中展示了两种不同模型在多个训练阶段的CTM的注意力模式、准确率和最高确定性点，覆盖了所有64个元素。注意力和确定性模式的证据表明，这些CTM正在利用不同的算法来解决累积对等任务。当使用100个内部时钟时，注意力从序列的开始移动到结束，随之而来的是模型对其预测位置的确定性增加。另一方面，使用75次迭代的CTM学会了以逆序方式关注序列，在最终的内部时钟期间同时准确预测序列中大多数元素的对等性。这种逆序的数据搜索表明，CTM正在执行某种形式的规划，在做出最终决策之前逐步构建对观察到的数据的理解。这些结果表明，尽管存在多种解决此任务的策略，其中一些策略比其他策略更具可解释性，但CTM显然展示了形成并遵循策略的能力。

8.2. Demonstrations

We show two demonstrations in Figure 21. The first example (top) depicts a typical sample from the dataset, which contains values of 1 and −1 at random positions. In this case, the CTM perfectly predicts the cumulative parity. The dynamics of the attention heads (a) shows that the attention moves sequentially through the input data, in agreement with Figure 20. Furthermore, we can see that some heads attend to only positive or negative values, while other heads attend to both. The second example (bottom) shows a failure case of the model. When presented with an input sequence containing only positive parities, the model struggles to accurately predict the cumulative parity, highlighting an edge-case limitation.



(a) Attention trajectories (b) Prediction (c) Target

(d) Attention trajectories (e) Prediction (f) Target

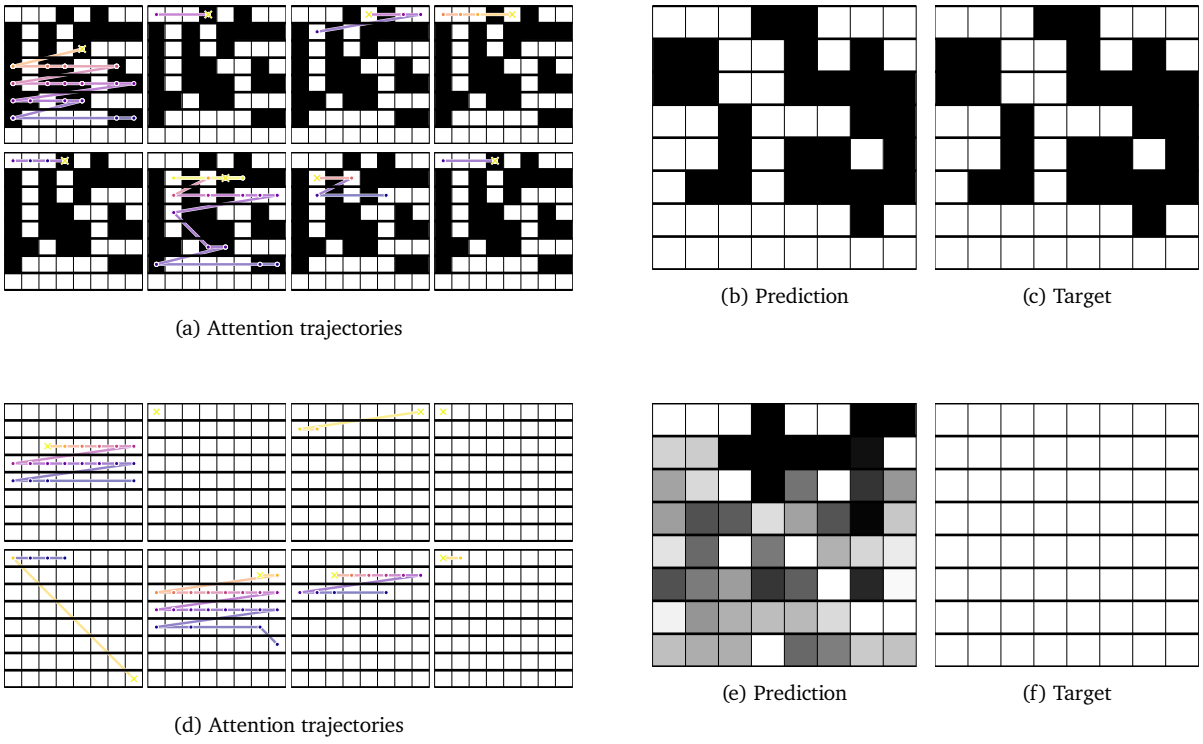
Figure 21 | Visualization of attention dynamics for two sample solutions to the parity task. The top row (a, b, c) depicts a perfect prediction, while the bottom row depicts a failure case (d, e, f). Attention trajectories (i.e., positions of the argmax in the attention weights) for each of the eight attention heads are shown in (a, d). Each point indicates the input position with the highest attention weight at a given timestep. Color represents progression over time, with brighter colors indicating later internal ticks. The cross (×) marks the timestep at which the model reached maximum certainty in its prediction. Attention heads typically move sequentially through input positions. Certain heads consistently attend only to positive or negative input values, whereas others alternate between positive and negative input values. Similarly, some heads remain relatively static, while others move quickly over the data. (b, e) The model's predictions. (c, f) The target.

9. Q&A MNIST

To assess the CTM's capabilities for memory, retrieval, and arithmetic computation, we devise a Question and Answering (Q&A) MNIST task, reminiscent of Manhaeve et al. (2018) or Schlag and Schmidhuber (2021). In this task, the model sequentially observes a series of MNIST digits (LeCun et al., 1998), followed by an interwoven series of index and operator embeddings that determine which of the observed digits to select and which modular operation to perform over them. This allows us to probe whether the CTM can simultaneously recognize hand-drawn digits, recall previous observations, and perform logical computation on them without any prior knowledge of the digits depicted in the images or the relationships between them. Furthermore, by applying more operations at inference time than observed during training time, we can test the generalizability of the CTM.

8.2. 展示

我们在图21中展示了两个演示。第一个示例（顶部）描绘了数据集中的一般样本，其中包含随机位置的1和−1的值。在这种情况下，CTM完美地预测了累积奇偶性。注意力头的动态（a）显示，注意力按顺序通过输入数据，与图20一致。此外，我们可以看到一些头只关注正数或负数，而其他头则同时关注两者。第二个示例（底部）展示了模型的一个失败案例。当输入序列仅包含正数奇偶性时，模型难以准确预测累积奇偶性，突显出一个边缘案例限制。



(a) Attention trajectories (b) Prediction (c) Target

(d) Attention trajectories (e) Prediction (f) Target

图 21 | 两个样本解对奇偶性任务的注意力动态可视化。顶部行 (a, b, c) 展示了一个完美的预测，而底部行展示了失败案例 (d, e, f)。每个注意力头（即注意力权重中的 argmax 位置）在每个时间步的注意力轨迹在 (a, d) 中显示。每个点表示给定时间步具有最高注意力权重的输入位置。颜色表示时间的进展，较亮的颜色表示较晚的时间步。交叉 (×) 标记了模型达到预测最大确定性的时间步。通常，注意力头按顺序通过输入位置。某些头始终只关注正输入值或负输入值，而其他头则在正输入值和负输入值之间交替。同样，某些头保持相对静止，而其他头则快速移动过数据。(b, e) 模型的预测。(c, f) 目标。

9. 常见问题 MNIST

为了评估CTM在记忆、检索和算术计算方面的能力，我们设计了一个基于MNIST的数据问答（Q&A）任务，类似于Manhaeve等人（2018）或Schlag和Schmidhuber（2021）的方法。在这个任务中，模型依次观察一系列MNIST数字（LeCun等人，1998），随后是一系列交织的索引和操作嵌入，这些嵌入决定了选择哪些观察到的数字以及在它们之间执行哪种模块化操作。这使我们能够探究CTM是否能够同时识别手写数字、回忆之前的观察，并在没有任何关于图像中或它们之间关系的先验知识的情况下对它们进行逻辑计算。此外，通过在推理时应用比训练时观察到的更多操作，我们可以测试CTM的泛化能力。

Specifically, the model first observes N_d MNIST digits sequentially for t_d internal ticks each. Next, the model receives an interwoven sequence of N_{idx} index embeddings (indicating which digit to select) and N_{op} operator embeddings (specifying either modular addition or subtraction, where each intermediate result is taken modulo 10 to keep answers within the range 0–9), each presented for t_{idx} and t_{op} internal ticks, respectively. Finally, the model observes a zero tensor for t_{ans} internal ticks, signaling the model to produce its answer. The target, between 0 and 9, results from the composition of all specified modular arithmetic operations. An example is shown in Figure 22.

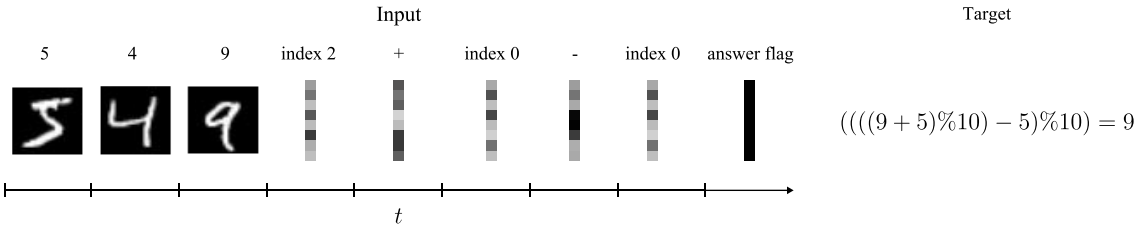


Figure 22 | Overview of the Q&A MNIST task. The model observes a series of digits followed by a series of index and operator embeddings, each repeated for several internal ticks. The model is then shown an answer flag and must predict the result of the modular operations.

We trained CTMs and parameter-matched LSTMs with two different configurations, varying how many internal ticks were used to process each input. Digits and embeddings were observed for either 1 or 10 internal ticks, with corresponding answering times of 1 or 10 internal ticks. The number of digits and the number of operations were sampled uniformly between 1 and 4. Memory lengths for the 1 and 10 internal ticks per input CTMs were set to 3 and 30 steps, respectively. We highlight that with these observation and memory length configurations that the digit observations will always lie outside of the memory length-sized window during the answering stage. In this way, the CTM must organize its activations such that it can recall the digits at later time steps. The CTM is trained with the loss defined in Section 2.5, computed only over the final t_{ans} steps. Once more, we set t_2 to be the final iteration for the LSTM for stable training. A full overview can be found in Appendix H.

9.1. Results

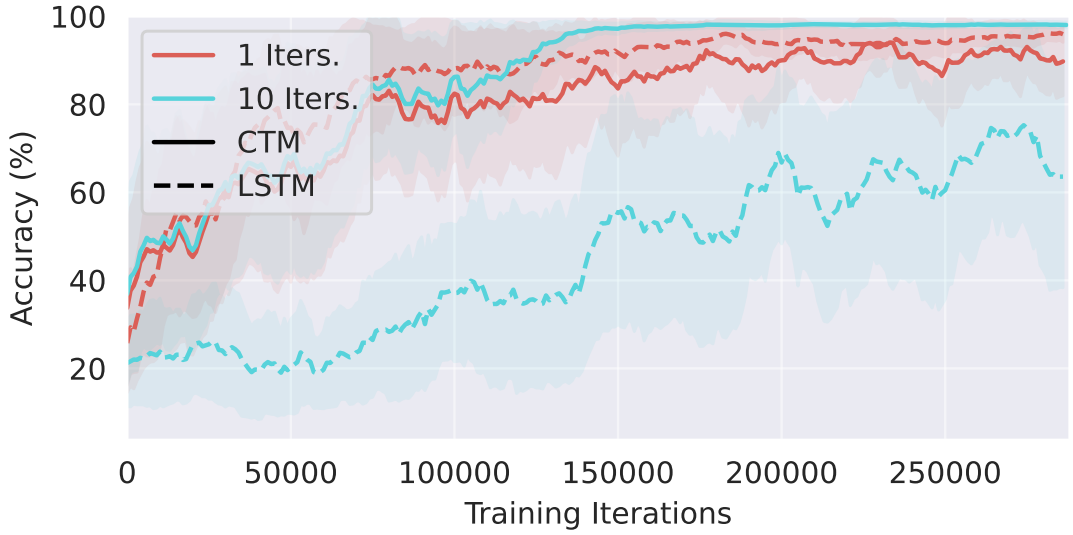


Figure 23 | Training curves for the CTM and LSTM on the Q&A MNIST task. The shaded areas represent one standard deviation across seeds. With a single internal tick, the LSTM outperforms the CTM. However, the performance of the CTM increases with the number of internal ticks, while the LSTM becomes increasingly unstable.

具体地，模型首先依次观察 N_d 个 MNIST 数字，每次 t_d 个内部时钟周期。接下来，模型接收一个交织的序列，包括 N_{idx} 个索引嵌入（指示选择哪个数字）和 N_{op} 个操作嵌入（指定模 10 加法或减法，每个中间结果取模 10 以保持在 0–9 范围内），分别在 t_{idx} 和 t_{op} 个内部时钟周期中呈现。最后，模型观察一个零张量 t_{ans} 个内部时钟周期，指示模型生成其答案。目标值（在 0 到 9 之间）是所有指定的模算术操作的组合结果。一个示例如图 22 所示。

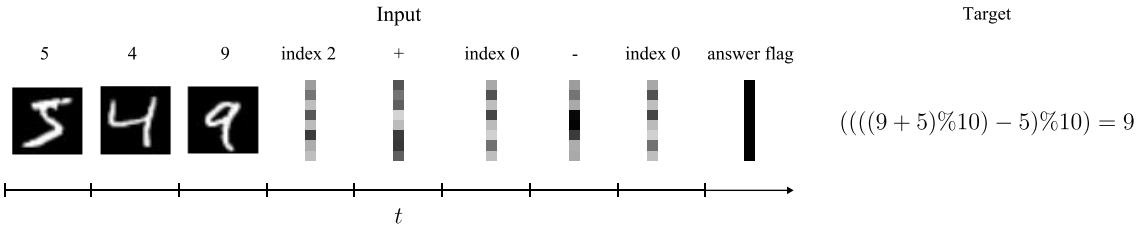


图22 | Q&A MNIST任务概述。模型观察一系列数字，随后是一系列索引和操作符嵌入，每种嵌入在多个内部时钟周期中重复出现。然后模型会看到一个答案标志，并必须预测模运算的结果。

我们使用两种不同的配置训练了CTMs和参数匹配的LSTMs，变化了用于处理每个输入的内部时钟数量。每个输入的内部时钟数量为1或10时，观察到的数字和嵌入对应于1或10个内部时钟。数字的数量和操作的数量在1到4之间均匀采样。对于每个输入具有1个内部时钟和10个内部时钟的CTMs，内存长度分别设置为3步和30步。我们强调，在这些观察和内存长度配置下，在回答阶段，数字观察值将始终位于内存长度大小的窗口之外。这样，CTM必须组织其激活，以便在后续时间步长中回忆这些数字。CTM使用第2.5节中定义的损失进行训练，并仅在最后的t_{ans}步中计算。再次，我们将t_2设置为LSTM的最终迭代，以实现稳定的训练。完整的概述可以在附录H中找到。

9.1. 结果

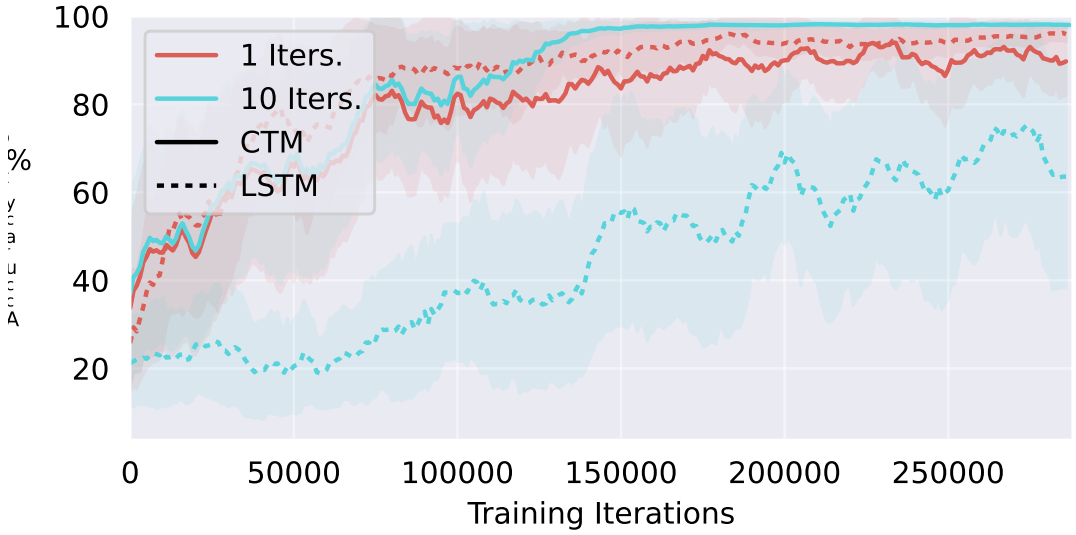
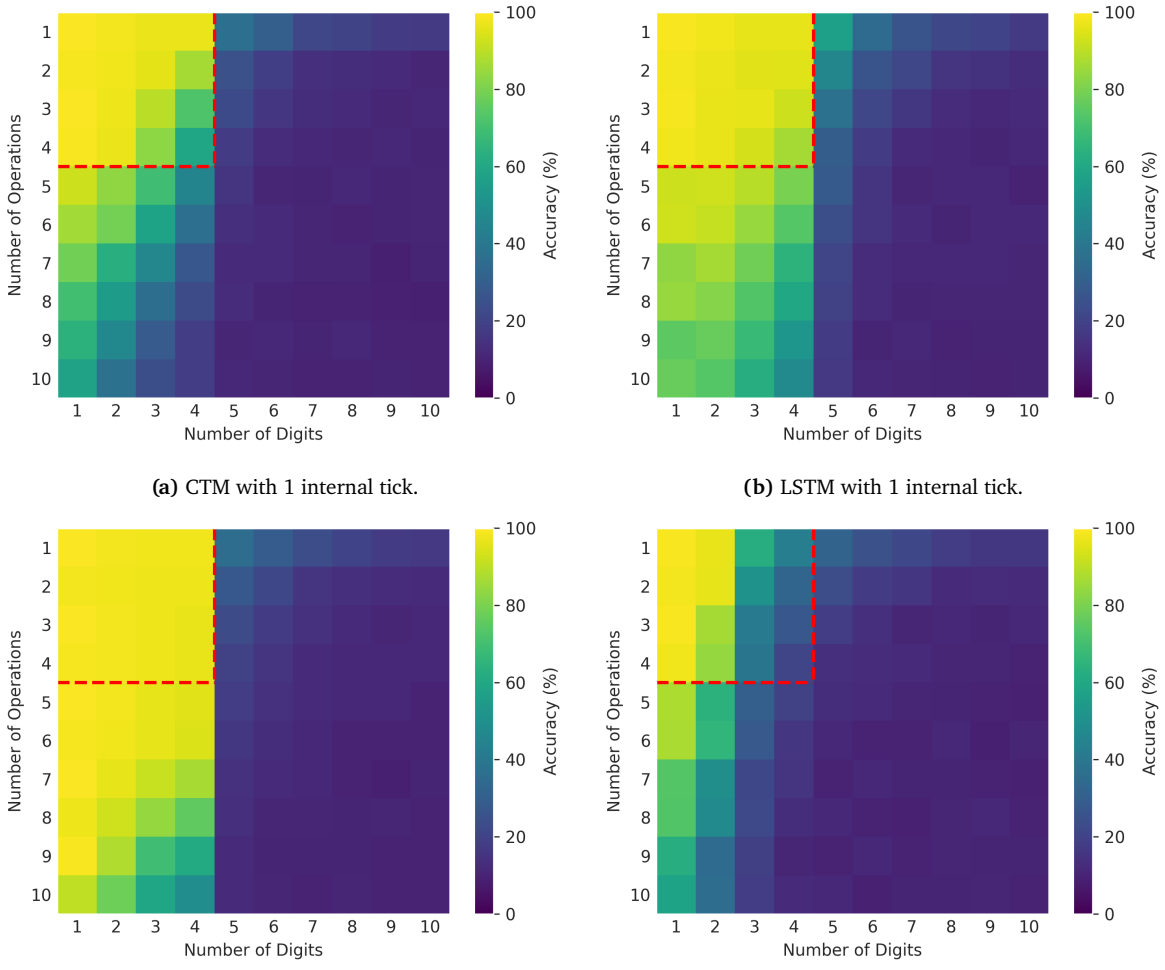


图 23 | CTM 和 LSTM 在 Q&A MNIST 任务中的训练曲线。阴影区域表示不同种子的一标准差。带有单个内部刻度的 LSTM 在性能上优于 CTM。然而，CTM 的性能随着内部刻度数量的增加而提高，而 LSTM 的性能则变得越来越不稳定。

Memory via synchronization. Training curves for three seeded runs for CTMs and parameter-matched LSTMs are shown in Figure 23. With a single internal tick, the LSTM initially outperforms the CTM. As the number of internal ticks increases, the LSTM's performance degrades and learning becomes considerably more unstable. In contrast, the CTM consistently improves its performance with additional thinking time. Specifically, all three seeded runs for the CTM with 10 internal ticks per input achieved over 96% accuracy on the most challenging in-distribution task (performing four operations after observing four digits). In contrast, the corresponding 10-internal tick LSTM performed at or below 21% accuracy across all seeded runs. The strong performance of the single-tick LSTMs highlights the effectiveness of the LSTM's complex gated update, however, this mechanism does not scale effectively to multiple internal steps, unlike the CTM, which effectively utilizes internal ticks to build up a synchronization representation.

The CTM performs well even when the observed digits are outside of the memory window, indicating that it has learned to memorize what it has observed to some degree, purely via the organization and synchronization of neurons. The strong performance of the CTM indicates that processing timing information through the synchronization of neuron activations may be a powerful mechanism for memorization and recall.



(a) CTM with 1 internal tick.

(b) LSTM with 1 internal tick.

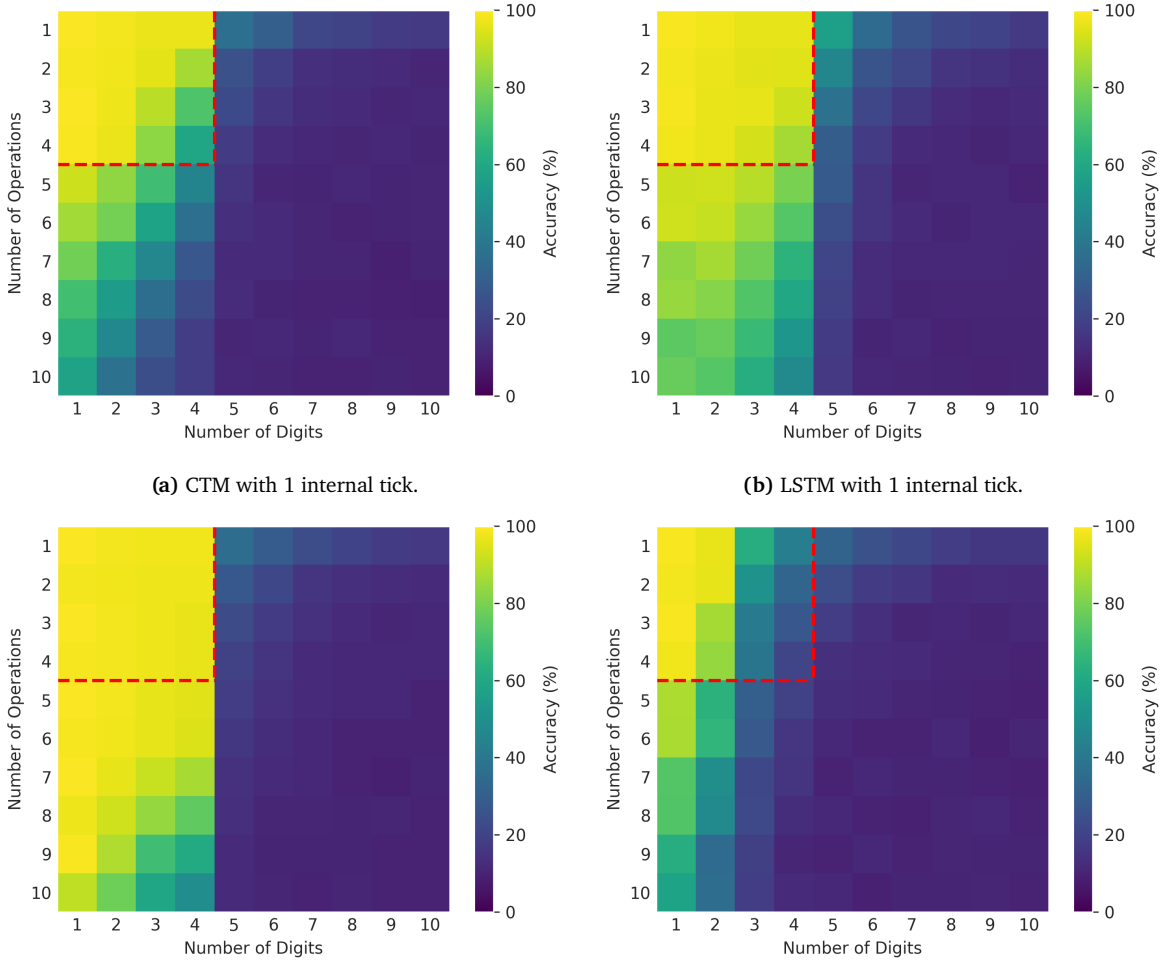
(c) CTM with 10 internal ticks.

(d) LSTM with 10 internal ticks.

Figure 24 | Generalizability on the Q&A MNIST task for CTM and LSTM models with 1 and 10 internal ticks. The x-axis denotes the number of MNIST digits input to the model, the y-axis denotes the number of operations the model must perform, and the color corresponds to the test accuracy.

通过同步实现记忆。图23显示了CTMs和参数匹配的LSTMs在三次种子运行中的训练曲线。在单个内部时钟周期下，LSTM最初优于CTM。随着内部时钟周期数量的增加，LSTM的性能下降，学习变得显著更加不稳定。相比之下，CTM在增加思考时间后持续提高其性能。具体来说，对于每个输入有10个内部时钟周期的CTM的三次种子运行，在最具有挑战性的内部分布任务（在观察到四个数字后执行四个操作）上均实现了超过96%的准确性。相比之下，对应的每内部时钟周期10次的LSTM在所有种子运行中的准确率均低于或等于21%。单个时钟周期LSTMs的出色表现突显了LSTM复杂门控更新机制的有效性，然而，这种机制并不像CTM那样能够有效地扩展到多个内部步骤，而CTM能够有效地利用内部时钟周期来构建同步表示。

CTM 在观察到的数字超出记忆窗口的情况下仍然表现良好，这表明它已经学会在一定程度上记住它所观察到的内容，纯粹是通过神经元激活的组织和同步实现的。CTM 的强大表现表明，通过神经元激活的同步处理时间信息可能是记忆和回忆的一种强大机制。



(a) CTM with 1 internal tick.

(b) LSTM with 1 internal tick.

(c) CTM with 10 internal ticks.

(d) LSTM with 10 internal ticks.

图24 | CTM和LSTM模型在Q&A MNIST任务上的泛化能力，使用1和10个内部时间刻度。x轴表示输入给模型的MNIST数字数量，y轴表示模型必须执行的操作数量，颜色对应于测试准确率。

The CTM can generalize. We examine generalization by measuring the accuracy of the models when given more digits or index-operator embeddings than used during training. Figure 24 shows the accuracy of the CTM and LSTM as a function of the number of digits shown and operations to perform, with the training regime highlighted in red. We find that both the CTM and LSTM baselines can generalize to an increased number of operations. To understand how the model is capable of generalizing out of distribution, we illustrate an example thought process of the CTM in Figure 25, which shows a sample sequence of inputs and a snapshot of the output logits. We find that the CTM sequentially computes the modular computation as the embeddings are observed, instead of waiting for the final answer flag to determine the final solution at once. A similar behavior can be seen in the 1-internal tick LSTM baseline. We are not claiming that the CTM can do something that the LSTM cannot, but instead that it can learn to **use synchronization as a tool** to solve this task, and that the result is both effective and scales to longer task requirements.

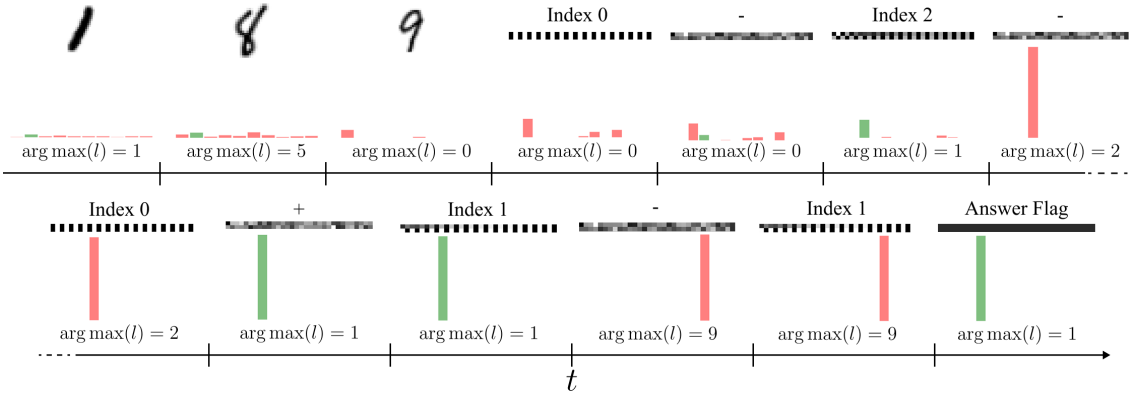
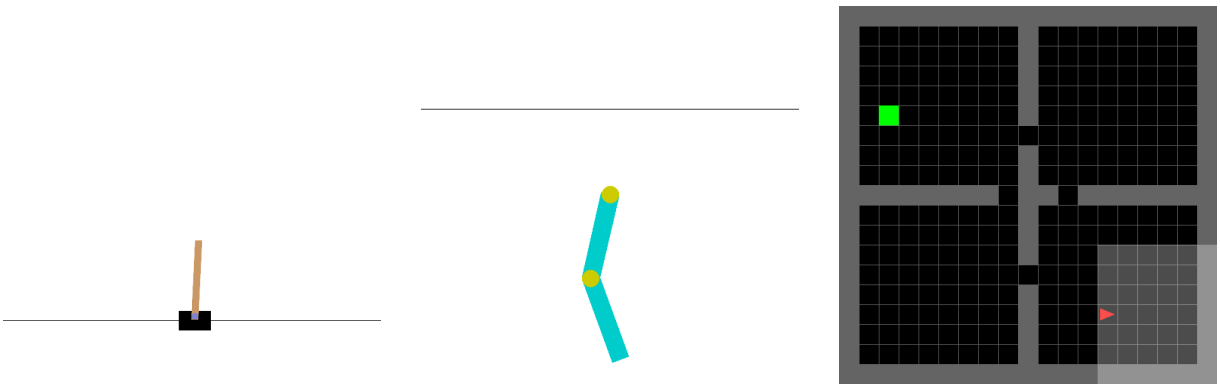


Figure 25 | Example CTM thought process from the Q&A MNIST task. Shown are the inputs to the model (MNIST digit, index and operator embeddings) as well as the argmax of the output logits l, at different snapshots. Each input is repeated for 10 internal ticks. In this case, the model is to compute ((((((1 − 9)%10) − 1)%10 + 8)%10 − 8)%10). We find that the model computes each part of this composition sequentially as the embeddings are observed, with outputs of 2, 1, 9, and finally, the correct answer of 1, projected from the synchronization representation.

10. Reinforcement learning



(a) CartPole **(b)** Acrobot **(c)** MiniGrid Four Rooms

Figure 26 | Reinforcement learning environments. In CartPole (a), the agent balances a pole on a cart using two discrete actions (left or right). Observations include two non-masked inputs: cart position and pole angle. In Acrobot (b), the agent applies one of three torques (+1, −1, or 0) to a two-joint arm to raise its end above a target height, using four non-masked inputs (sines and cosines of joint angles). In MiniGrid Four Rooms (c), the agent navigates using seven discrete actions (e.g., turn left, turn right, etc.) and observes a $3 \times 7 \times 7$ input tensor encoding object, color, and state IDs within a 7×7 limited field of view.

CTM 可以泛化。我们通过测量在训练时提供的数字或索引操作嵌入更多时模型的准确性来检查泛化能力。图24显示了CTM和LSTM的准确性随显示的数字数量和需要执行的操作数量的变化情况，并用红色高亮显示了训练制度。我们发现，无论是CTM还是LSTM基线，都可以泛化到更多的操作数量。为了理解模型如何能够超出训练数据范围进行泛化，我们在图25中展示了CTM的一个示例思维过程，该图显示了一个输入序列样本和输出logits的一个快照。我们发现，当嵌入被观察到时，CTM会顺序计算模块化计算，而不是等待最终答案标志来一次性确定最终解决方案。1-内部时钟LSTM基线也有类似的行为。我们并不声称CTM可以做LSTM不能做的事情，而是说它可以学会使用同步作为一种工具来解决这个问题，并且结果既有效又可以扩展到更长的任务需求。

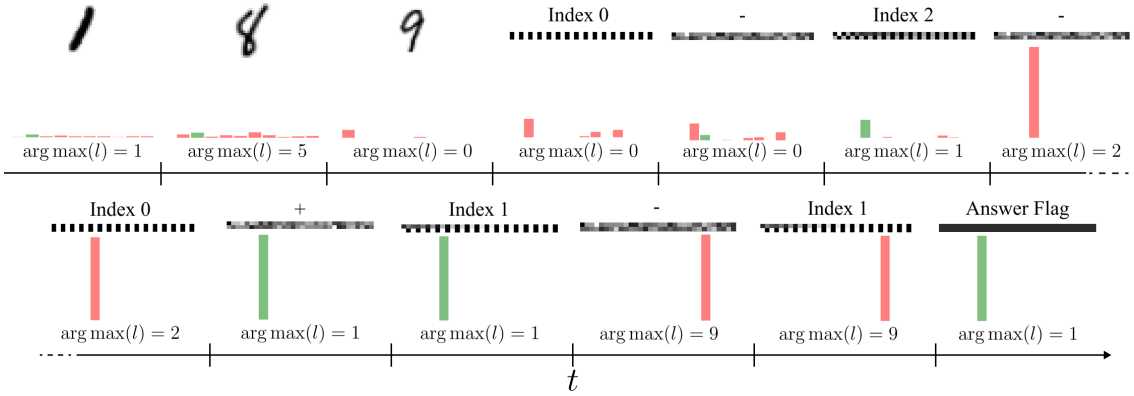
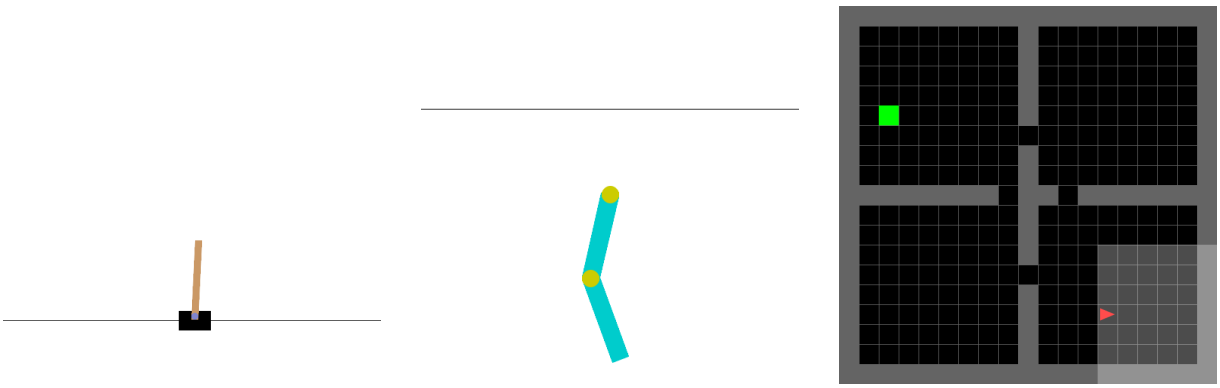


图 25 | Q&A MNIST 任务的 CTM 思维过程示例。展示了模型的输入（MNIST 数字、索引和操作嵌入）以及输出 logits 的 argmax 在不同时间点的结果。每个输入在内部时钟上重复了 10 次。在这种情况下，模型需要计算 ((((((1 − 9)%10) − 1)%10 + 8)%10 − 8)%10)。我们发现，当嵌入被观察到时，模型会顺序计算这个组合的每一部分，输出分别为 2、1、9，最后从同步表示中投影出正确的答案 1。

10. 强化学习



(a) CartPole **(b)** Acrobot **(c)** MiniGrid Four Rooms

图 26 | 强化学习环境。在 CartPole (a) 中，智能体使用两个离散动作（左或右）使杆平衡在小车上。观察包括两个非遮蔽输入：小车位置和杆角度。在 Acrobot (b) 中，智能体对一个两关节臂施加一个扭矩（+1、−1 或 0），以使其末端高于目标高度，使用四个非遮蔽输入（关节角度的正弦和余弦）。在 MiniGrid 四个房间 (c) 中，智能体使用七个离散动作（例如，向左转、向右转等）导航，并观察一个 3×7×7 输入张量，该张量编码视野内 7×7 的物体、颜色和状态 ID。

We have previously shown that the CTM can process sequentially on non-sequential tasks via its decoupled internal recurrence. Here, we extend the CTM to sequential decision-making tasks involving interactions with external environments. Specifically, we train CTMs using reinforcement learning (RL), where the model learns action-selection policies based on environmental observations and trial-and-error interactions. In this setting, the CTM processes one or more internal ticks before producing an action that transitions the environment to the next state. To achieve this, we continuously maintain the neuron dynamics across these internal ticks over successive environment steps, allowing previous environmental observations to influence current internal states via the NLMs. A central goal of this section is to provide evidence that the CTM can be set up to learn in a continuous environment.

Environments. We test the CTM on two classic control tasks and one navigation task, namely, CartPole, Acrobot and MiniGrid Four Rooms, implemented in Gymnasium (Barto et al., 1983; Chevalier-Boisvert et al., 2023; Sutton, 1995; Towers et al., 2024). Examples of these tasks are shown in Figure 26. Because the CTM maintains an activation history across environment transitions, it functions as a stateful recurrent neural network. Therefore, we specifically evaluate the CTM in partially observable settings, where RNNs are effective (Hausknecht and Stone, 2015). Partial observability is introduced by masking the positional and angular velocity observation components in the control tasks and restricting the field of view in the navigation task. This masking converts these tasks into partially observable Markov decision processes (POMDPs), requiring the CTMs to develop policies that recall past observations. For instance, in the Acrobot task, selecting the correct action depends on recalling past positions and inferring the velocity to increase arm elevation.

Architecture. For the RL tasks, the following architecture is used and trained with Proximal Policy Optimization (Schulman et al., 2017). First, the input observations are processed using a series of fully connected layers. For the navigation task, this also includes embedding the observed states and adding positional embeddings, corresponding to locations within the agent's field of view. This representation is then processed by the CTM, without the use of an attention mechanism, for a number of internal ticks, before the synchronization vector is output and processed by the actor and critic heads. We compare this approach with parameter-matched LSTMs baselines, where the internal ticks are instead processed by an LSTM cell, which outputs its hidden state to the actor and critic networks. The purpose of such comparison is not too showcase a superiority of one architecture over another, but to show that a CTM can leverage the synchronization of a continuous history of activations, and achieve a comparable performance to LSTMs. A full description of the architecture and optimization hyperparameters can be found in Appendix I.

10.1. Results

The CTM can continuously interact with the world. Training curves for the reinforcement learning tasks are shown in Figure 27. In all tasks, we find that the CTM achieves a similar performance to the LSTM baselines.

Figure 28 compares the neuron traces of the CTM and the LSTM baselines for the CartPole, Acrobot and MiniGrid Four Rooms tasks. In the classic control tasks, the activations for both the CTM and the LSTM feature oscillatory behavior, corresponding to the back-and-forth movements of the cart and arm. For the navigation task, a rich and complex activation pattern emerges in the CTM. The LSTM on the other hand, features a less diverse set of activations. The LSTMs trained in this section have a more dynamic neural activity than what can be seen when trained on CIFAR-10 (Figure 12). This is likely due to the sequential nature of RL tasks, where the input to the model changes over time owing to its interaction with the environment, inducing a feedback loop that results in the model's latent representation also evolving over time.

我们之前已经证明，CTM可以通过其解耦的内部递归处理顺序任务中的非顺序任务。在这里，我们将CTM扩展到涉及与外部环境交互的顺序决策任务。具体来说，我们使用强化学习（RL）训练CTM，其中模型根据环境观察和尝试-错误交互学习动作选择策略。在这种设置中，CTM在产生一个动作之前会处理一个或多个内部时钟，该动作将环境过渡到下一个状态。为了实现这一点，我们在连续的环境步骤中持续维护神经元动力学，使得先前的环境观察能够通过NLMs影响当前的内部状态。本节的一个主要目标是提供证据，证明CTM可以设置为在连续环境中学习。

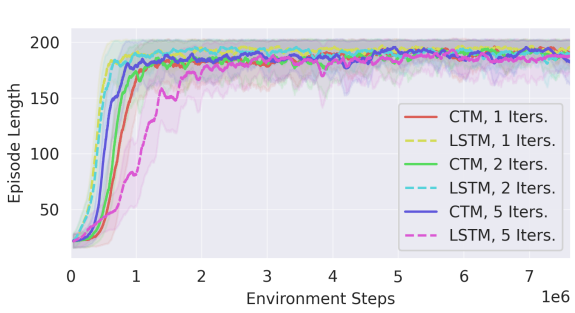
环境。我们在两个经典的控制任务和一个导航任务上测试了CTM，分别是Cart-Pole、Acrobot和MiniGrid Four Rooms，这些任务在Gymnasium中实现（Barto等人，1983；Chevalier-Boisvert等人，2023；Sutton，1995；Towers等人，2024）。这些任务的示例如图26所示。由于CTM在环境转换中保持激活历史记录，它作为有状态的递归神经网络发挥作用。因此，我们特别在部分可观测设置中评估CTM，这是RNNs有效发挥作用的场景（Hausknecht和Stone，2015）。部分可观测性通过在控制任务中遮蔽位置和角速度观测组件以及在导航任务中限制视野引入。这种遮蔽将这些任务转换为部分可观测马尔可夫决策过程（POMDPs），要求CTMs开发能够回忆过去观测的策略。例如，在Acrobot任务中，选择正确的动作取决于回忆过去的姿态并推断速度以增加臂部抬升。

架构。对于RL任务，使用了以下架构，并使用 proximal policy optimization (Schulman等，2017) 进行训练。首先，输入观察值通过一系列全连接层进行处理。对于导航任务，这还包括将观察到的状态嵌入，并添加位置嵌入，对应于代理视野内的位置。然后，此表示由CTM处理，不使用注意力机制，经过一定数量的内部时钟周期后，输出同步向量并由行为头和批评头处理。我们将这种方法与参数匹配的LSTM基线进行比较，在这种情况下，内部时钟周期由LSTM单元处理，LSTM单元将其隐藏状态输出给行为网络和批评网络。这种比较的目的是展示一种架构并不一定优于另一种架构，而是展示CTM可以利用连续激活历史的同步，并达到与LSTM相当的性能。架构和优化超参数的完整描述可以在附录I中找到。

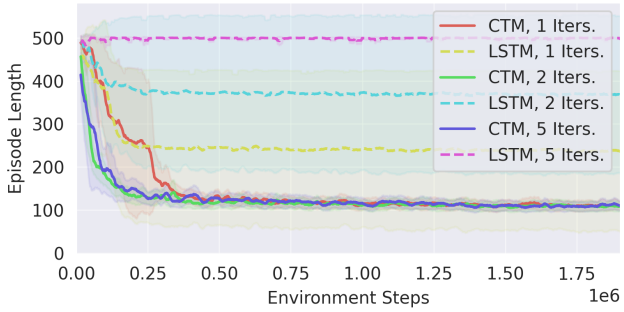
10.1. 结果

T他可以连续与世界交互。强化学习的训练曲线
t询问如图27所示。在所有任务中，我们发现CTM在性能上与{v*}相当。
LSTM基线。

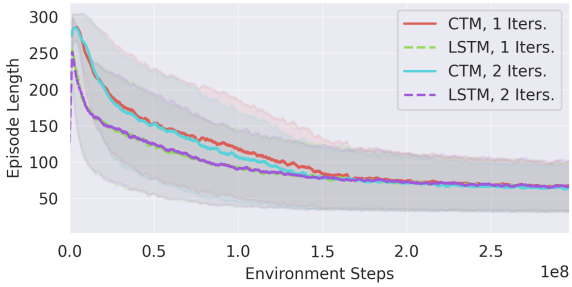
图28比较了CTM和LSTM基线在CartPole、Acrobot和MiniGrid Four Rooms任务中的神经元轨迹。在经典的控制任务中，CTM和LSTM的激活行为都表现出振荡特性，对应于小车和臂的前后运动。对于导航任务，CTM出现了丰富而复杂的激活模式。另一方面，LSTM的激活模式则较为单一。本节训练的LSTMs具有比在CIFAR-10（图12）上训练时更动态的神经活动。这可能是因为强化学习任务的序列特性，模型的输入会随着时间与其环境的交互而变化，从而产生反馈循环，导致模型的潜在表示随着时间的推移而演变。



(a) Cartpole

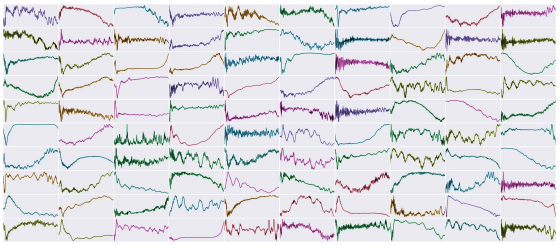


(b) Acrobot

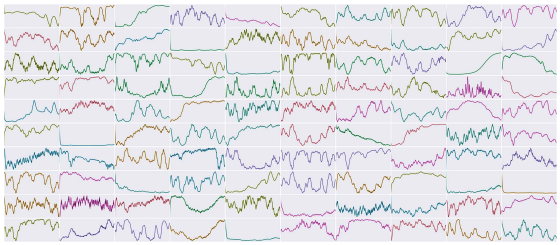


(c) MiniGrid 4-Rooms

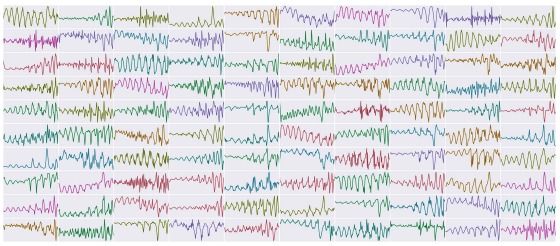
Figure 27 | Training curves for reinforcement learning tasks. Each curve depicts a moving average of the episode length during training, averaged over three training runs. The shaded region represents one standard deviation across seeds. For Cartpole, higher is better. For Acrobot and MiniGrid 4-rooms, lower is better.



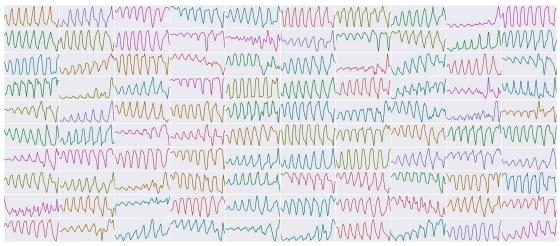
(a) CTM, CartPole, 2 Internal Ticks



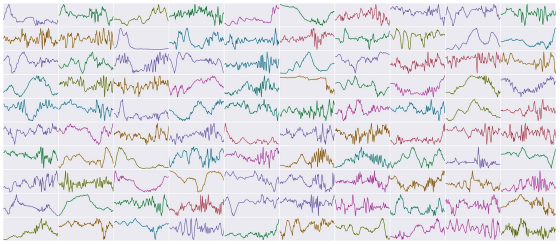
(b) LSTM, CartPole, 2 Internal Ticks



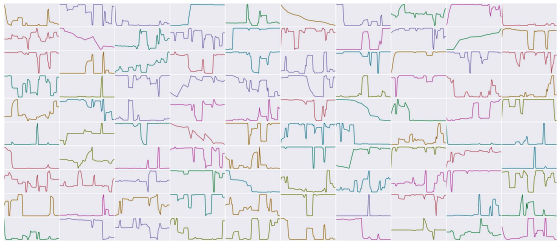
(c) CTM, Acrobot, 1 Internal Tick



(d) LSTM, Acrobot, 1 Internal Tick

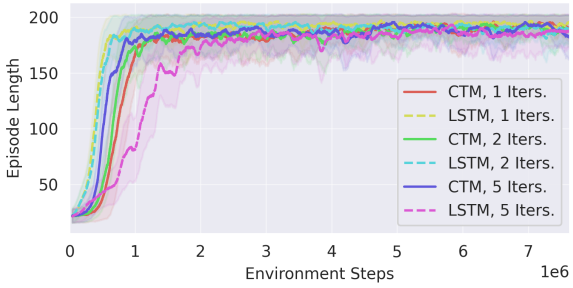


(e) CTM, MiniGrid Four Rooms, 2 Internal Ticks

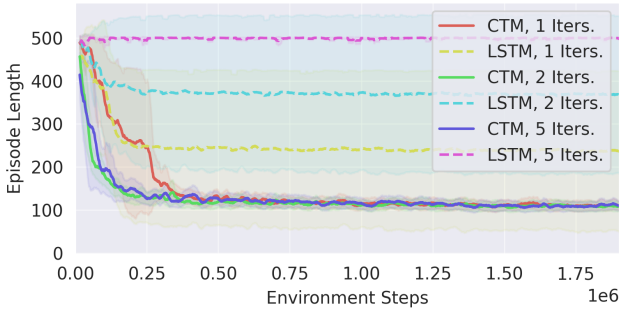


(f) LSTM, MiniGrid Four Rooms, 2 Internal Ticks

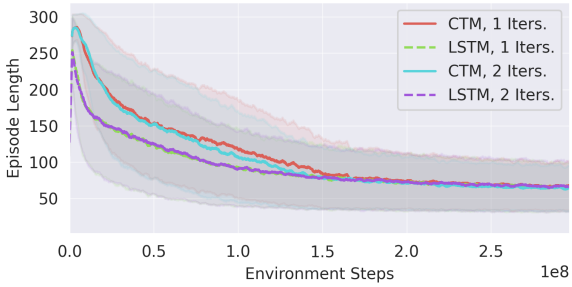
Figure 28 | Neural activities over the course of a single episode for the CTM and LSTM on CartPole, Acrobot and MiniGrid Four Rooms tasks. The CTM features richer neuron dynamics than the LSTM.



(a) Cartpole

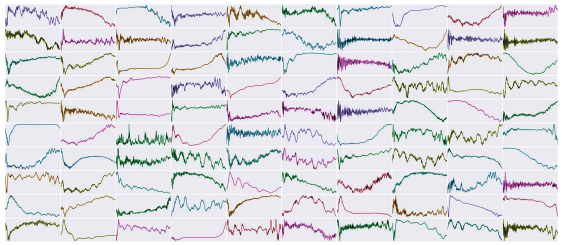


(b) Acrobot

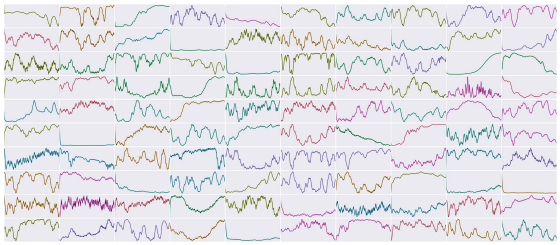


(c) MiniGrid 4-Rooms

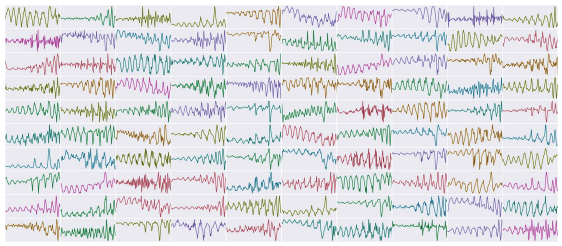
F图 27 | 强化学习任务的训练曲线。每条曲线表示episode长度的移动平均值
d在训练过程中，平均值基于三次训练运行。阴影区域表示不同种子的标准偏差范围。对于
Cartpole, 越高越好。对于Acrobot和MiniGrid 4-rooms，越低越好。



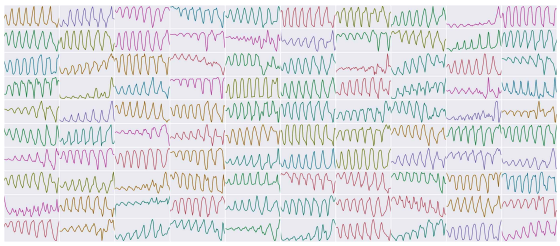
(a) CTM, CartPole, 2 Internal Ticks



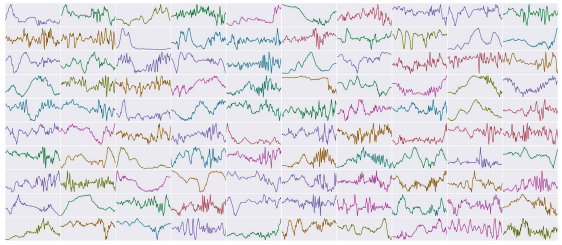
(b) LSTM, CartPole, 2 Internal Ticks



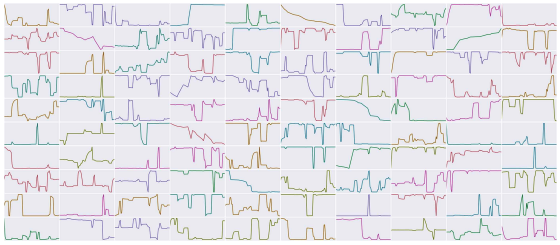
(c) CTM, Acrobot, 1 Internal Tick



(d) LSTM, Acrobot, 1 Internal Tick



(e) CTM, MiniGrid Four Rooms, 2 Internal Ticks



(f) LSTM, MiniGrid Four Rooms, 2 Internal Ticks

图28 | 在CartPole、Acrobot和MiniGrid 四个房间任务中，CTM和LSTM在整个单个episode中的神经活动。CTM的神经元动力学比LSTM更丰富。

11. Related work

Modern neural networks have achieved remarkable success across a range of domains, yet they typically rely on fixed-depth, feedforward computation, with limited flexibility to adapt their processing based on input complexity. In contrast, biological brains exhibit dynamic neural activity that unfolds over time, adjusting computation to the demands of a task. The CTM builds on this idea by explicitly modeling internal neural timing and synchronization. In this section, we highlight key works related to *adaptive computation*, *iterative reasoning*, and *biologically inspired architectures*, all of which inform the motivation behind the CTM.

11.1. Adaptive computation and dynamic halting

A number of approaches have explored *adaptive computation*, where the number of inference steps varies based on input difficulty or confidence. Early-exit networks (e.g., Bolukbasi et al. (2017)) allow models to terminate inference early if intermediate layers produce confident predictions, thereby saving computation on easy examples. PonderNet (Banino et al., 2021) introduced a stochastic halting mechanism for recurrent models, enabling learned per-input "ponder times" through an end-to-end differentiable loss that balances accuracy and efficiency. This method improved upon Adaptive Computation Time (ACT) (Graves, 2016) by offering more stable training and stronger generalization on algorithmic reasoning tasks.

More recently, AdaTape (Xue et al., 2023) proposed a flexible memory-augmented architecture that dynamically extends the input with additional "tape tokens," effectively granting the model more computation budget as needed. Similarly, the Sparse Universal Transformer (SUT) (Tan et al., 2023) combines recurrent weight sharing with dynamic halting and Mixture-of-Experts routing, enabling the model to apply varying numbers of recurrent transformer layers per input. These methods demonstrate the benefits of input-dependent computation, aligning compute cost with problem difficulty—a goal also pursued by the CTM via its internal "thought" dimension.

11.2. Iterative and recurrent reasoning

The CTM shares common ground with models designed for *iterative reasoning* and *internal recurrence*. Quiet-STaR (Zelikman et al., 2024), for example, teaches language models to "think before speaking" by inserting hidden rationale tokens during training, encouraging internal computation before output generation. This process enhances performance on complex tasks like math reasoning and commonsense QA. Other architectures like Recurrent Independent Mechanisms (RIMs) (Goyal et al., 2019) split computation across sparsely activated, modular sub-networks, which evolve asynchronously over time, improving systematic generalization and multi-step reasoning. These approaches echo the CTM's aim of simulating thought through internal, decoupled computation that is not directly tied to the input sequence.

11.3. Biologically inspired neural dynamics

A growing body of work aims to make neural computation more biologically plausible (Schmidgall et al., 2024). Liquid Time-Constant Networks (LTCNs) (Hasani et al., 2021) use neurons governed by time-varying differential equations, enabling each neuron to adapt its response dynamically based on input history. These networks have shown strong performance on temporal tasks with high sample efficiency and robustness. Similarly, Spiking Neural Networks (SNNs) have gained traction as biologically grounded alternatives to standard deep networks, relying on discrete spikes and precise timing to encode and process information. Recent advances (e.g., Stan and Rhodes (2024)) combine SNNs with state-space models and synchronization mechanisms, achieving competitive or even superior performance on long-range sequence tasks.

The CTM draws inspiration from such neural mechanisms – particularly *temporal coding* and *neural*

11. 相关工作

现代神经网络在多个领域取得了显著的成功，但它们通常依赖于固定深度的前馈计算，缺乏根据输入复杂度灵活调整处理过程的能力。相比之下，生物大脑表现出动态的神经活动，这种活动会随着时间展开，并根据任务需求调整计算。CTM正是基于这一理念，明确地建模了内部神经的时序和同步。在本节中，我们强调了与*adaptive computation*、*iterative reasoning*和*biologically inspired architectures*相关的关键工作，这些工作为CTM的动机提供了指导。

11.1. 自适应计算和动态停止

多种方法探索了 *adaptive computation*，其中推理步骤的数量根据输入的难度或置信度而变化。早期退出网络（例如 Bolukbasi 等人 (2017)）允许模型在中间层产生自信预测时提前终止推理，从而在处理简单示例时节省计算量。PonderNet（Banino 等人，2021）为递归模型引入了一种随机停止机制，通过端到端可微损失平衡准确性和效率，从而为每个输入学习“思考时间”。该方法在算法推理任务上的训练更加稳定，并且泛化能力更强，优于 Graves (2016) 提出的自适应计算时间 (ACT) 方法。

最近，AdaTape（Xue等，2023）提出了一种灵活的记忆增强架构，该架构动态地将额外的“磁带标记”扩展到输入中，有效地在需要时为模型提供更多计算预算。类似地，稀疏通用变换器（SUT）（Tan等，2023）结合了递归权重共享与动态停止和专家混合路由，使模型能够根据输入的不同应用不同数量的递归变换器层。这些方法展示了输入依赖计算的好处，将计算成本与问题难度对齐——这也是通过CTM内部的“思考”维度追求的目标。

11.2. 迭代和递归推理

CTM 与设计用于 *iterative reasoning* 和 *internal recurrence* 的模型有着共同之处。例如，Quiet-STaR（Zelikman 等人，2024）通过在训练过程中插入隐藏的推理标记来教导语言模型“先思考后说话”，从而鼓励在输出生成之前进行内部计算。这一过程增强了在数学推理和常识问答等复杂任务上的性能。其他架构如递归独立机制（RIMs）（Goyal 等人，2019）将计算分布在稀疏激活的模块化子网络中，这些子网络随着时间的推移异步演化，从而提高系统的泛化能力和多步推理能力。这些方法反映了CTM通过内部解耦计算模拟思考的宗旨，这种计算不直接与输入序列相关联。

11.3. 生物启发的神经动力学

一项不断增长的研究旨在使神经计算更具生物可行性（Schmidgall等，2024）。液态时间常数网络（LTCNs）（Hasani等，2021）使用由时间变化的微分方程控制的神经元，使每个神经元能够根据输入历史动态调整其响应。这些网络在高样本效率和鲁棒性方面表现出强大的性能，特别是在时间任务上。类似地，脉冲神经网络（SNNs）因其基于标准深度网络的生物基础而受到关注，依赖于离散的脉冲和精确的时间编码和处理信息。最近的进展（例如，Stan和Rhodes（2024））将SNNs与状态空间模型和同步机制结合，实现了在长序列任务上具有竞争力甚至更优的性能。

The CTM 从这样的神经机制中汲取灵感 – 特别是 *temporal coding* 和 *neural*

synchrony – to encode information in the timing of activity rather than static activations. Unlike prior models, however, the CTM uses these dynamics directly as a latent representation for attention and output generation, leading to a unified architecture where reasoning emerges naturally from the interplay of evolving neuron states.

12. Discussion and future work

In this technical report we presented the CTM as a model that unfolds and leverages the temporal dynamics of neural activity as the primary mechanism for its intelligence. To our best knowledge, using **neural synchronization over time as the latent representation** for a model has never been accomplished, particularly at this scale. The CTM is an instantiation of a model that uses the precise interplay and timing of temporal dynamics – which is believed to be crucial in natural cognition – to successfully perform a diverse range of tasks, for which we gave evidence herein.

Our goal for this work was to introduce this new model, and the perspective that **neural dynamics can be a powerful tool for neural computation**. We hope that the research community can adopt aspects of our work to build more biologically plausible and performant AI. In the following subsections we offer some discussion and perspectives based on our observations.

12.1. Intuitive perspective, biological plausibility

The CTM's outputs, $\mathbf{y}^t$, are computed as linear projections from synchronization. Consider, for example, object classification, where $\mathbf{y}^t$ represents class predictions. In this scenario, the CTM must observe abstract features in the input data to produce specific **activation dynamics** within its neurons. These activation dynamics, in turn, must synchronize in a precise manner to generate accurate predictions. Intuitively, this implies that the CTM learns to develop **persistent patterns of neural activity** over internal ticks in response to the input data, effectively building up its output through a temporal process. This concept aligns with recent notions of reasoning and is a key motivation behind our choice of the term 'thought.' Furthermore, such a dynamic and temporal representation contrasts sharply with existing methods that use standard representations. While the experiments in this paper represent an initial exploration of the potential of this type of representation, its closer resemblance to biological processes suggests that its eventual utility could be substantial.

12.2. The strengths of synchronization over time

Multi-resolution. Our measurement of synchronization depends on activity over time, but not exactly on time itself. This potentially frees up some portion of synchronization to change slowly over time, while our mechanism for learnable decaying time dependency (see Section 2.4) enables the emergence of short-term dependence. The result is that synchronization is a representation that can capture events or perspectives at any number of resolutions. Many real-world scenarios present themselves as situations where features or ideas where such a multi-resolution perspective could be powerful; we will explore this in future work.

Memory Further to this point is that a given synchronization captures not only the CTM's cognition over some time period, but also the consequence of it taking action, owing to the recurrent cycle of internal ticks. Such a perspective is far more akin to an 'experience' than what a snapshot representation could yield. Hence, synchronization matrices as memories presents an interesting avenue for future work.

Plasticity and gradient-free learning. As we have defined it in this technical report, synchronization measures how neurons fire together, which is a very Hebbian-like perspective (Hebb, 2005; Najarro

synchrony – 使用活动的时间性来编码信息，而不是静态激活。然而，与之前的模型不同，CTM 直接将这些动态作为注意力和输出生成的潜在表示，从而形成一个统一的架构，在这种架构中，推理自然地源自不断演变的神经元状态的相互作用。

12. 讨论与未来工作

在本技术报告中，我们提出了CTM作为一种模型，它通过展开和利用神经活动的时序动态来实现其智能的主要机制。据我们所知，使用时序神经同步作为模型的潜在表示尚未实现，尤其是在这种规模上。CTM是利用时序动态的精确相互作用和时间——这被认为在自然认知中至关重要——来成功执行一系列任务的一个模型实例，我们在此提供了相应的证据。

我们的目标是介绍这种新模型，并且神经动力学可以成为神经计算的强大工具这一视角。我们希望研究社区能够采纳我们工作中的某些方面，以构建更加生物合乎实际且性能更佳的AI。在接下来的子部分中，我们将根据我们的观察提供一些讨论和视角。

12.1. 直观视角，生物可行性

CTM的输出y^t是从同步中计算出的线性投影。例如，在对象分类中，y^t代表类别的预测。在这种情况下，CTM必须观察输入数据中的抽象特征，以在其神经元中产生特定的激活动态。这些激活动态反过来必须以精确的方式同步，以生成准确的预测。直观上来说，这意味着CTM学会在面对输入数据时，发展出持久的神经活动模式，并通过时间过程有效地构建其输出。这一概念与最近的推理概念相吻合，也是我们选择使用“思考”一词作为术语的关键动机。此外，这种动态和时间上的表示与现有使用标准表示的方法形成了鲜明对比。虽然本文中的实验代表了这种表示潜在能力的初步探索，但其更接近于生物过程的事实表明，其最终的实用性可能是相当大的。

12.2. 时间同步的优势

多分辨率。我们对同步性的测量依赖于时间内的活动，但并不精确依赖于时间本身。这有可能释放出一部分同步性，使其随着时间缓慢变化，而我们用于可学习衰减时间依赖性的机制（参见第2.4节）则使短期依赖性的出现成为可能。结果是，同步性是一种可以捕捉任意分辨率事件或视角的表示。许多现实世界的情景可以被视为在这种多分辨率视角下可能非常强大的情况；我们将在未来的工作中探讨这一点。

Memory 到此为止，给定的同步不仅捕获了CTM在某个时间段的认知，还捕获了其采取行动的后果，这归因于内部时钟的反复循环。这种视角与“经验”更为接近，而不仅仅是快照表示所能提供的。因此，同步矩阵作为记忆为未来的工作提供了一个有趣的途径。

P弹性与无梯度学习。如我们在本技术报告中所定义的，同步
m衡量神经元如何共同放电，这是一种非常类似希伯的观点（Hebb, 2005; Najarro

and Risi, 2020). Leveraging this notion to explore lifelong learning (Kudithipudi et al., 2022; Wang et al., 2024), plasticity, or even gradient-free optimization are exciting avenues for future work.

Cardinality The full synchronization in a D dimensional CTM is $(D \times (D+1))/2$ dimensional (the upper triangle of the full synchronization matrix). Research suggests that large representations are advantageous for a number of reasons (Allen-Zhu et al., 2019; Frankle and Carbin, 2018). Synchronization gives us access to a large and meaningful representation space at no additional cost. We intend to explore what utility can be gained from such a high-cardinality representation space, particularly in the realm of multi-modal modeling.

12.3. Continuous worlds

We took inspiration from nature to build the CTM, but used well-established protocols and datasets when training it. These datasets and protocols, however, are not necessarily natural. For instance, independent and identically distributed data is expected when training conventional NNs, but that is not how the real-world operates. Events happen over time and are usually arranged accordingly. Hence, we wish to move toward training the CTM in a biologically plausible fashion for future work. Application to sequential data (e.g., videos, text), particularly when sampled in order during training, is a promising avenue for future work.

Language modeling. We have yet to apply the CTM to the task of language modeling, but it is straightforward to adapt it to ingesting and thinking about text given that it uses attention. Moreover, since it can build a world model and navigate it (see Section 4), the CTM could potentially do without positional encodings, instead building a contextualized 'world model' of what it observes. One potential avenue we encourage readers to explore is that of applying the CTM to pretrained language models. For future work, we will build and train custom CTMs on text data in order to understand its capability in that domain.

12.4. What failed, and how did we get here?

We believe it is important to explain what we initially tried as the core representation for the CTM (as opposed to synchronization). The activated latent space, $\mathbf{z}^t$, is an obvious candidate. We found, however, that the dynamic and complex nature of $\mathbf{Z}$ meant we needed to perform some smoothing operations (e.g., accumulating some 'holder' latent or logits). Further, given that this representation is strongly coupled to t, we found that the CTM would learn when exactly to produce an output (i.e., when the loss function was applied) instead of relying on the internal self-organization as the primary driving force.

By adding neuron timing back into the system we introduced this challenge; thankfully, synchronization, being time-independent, proved an elegant solution to overcoming these challenges. The learnable decaying time-dependence also offers a neat solution whereby the CTM can learn to interact with its world based on short-term neural behavior, should this be preferred.

13. Conclusion

The Continuous Thought Machine (CTM) represents a novel step towards bridging computational efficiency with biological plausibility in artificial intelligence. By moving beyond traditional pointwise activation functions to private neuron-level models, the CTM cultivates far richer neuron dynamics. Crucially, it leverages neural synchronization as a powerful and fundamentally new type of representation – distinct from the activation vectors prevalent since the early days of neural networks. This direct use of neuron dynamics as a first-class representational citizen allows the CTM to exhibit behaviors qualitatively different from contemporary models.

and Risi, 2020). 利用这一概念探索终身学习（Kudithipudi等，2022；Wang et al., 2024），或者甚至是无梯度优化，都是未来工作令人兴奋的途径。

基数 在一个 D 维的 CTM 中的全同步是 $(D \times (D+1))/2$ 维的（全同步矩阵的上三角部分）。研究表明，大型表示对于多种原因来说是有优势的（Allen-Zhu 等人，2019；Frankle 和 Carbin，2018）。同步使我们能够以零附加成本访问一个庞大且有意义的表示空间。我们打算探索这种高基数表示空间能带来什么样的实用性，特别是在多模态建模领域。

12.3. 连续世界

我们从自然界中汲取灵感构建了CTM，但在训练时使用了成熟的协议和数据集。然而，这些数据集和协议未必是自然的。例如，当训练传统神经网络时，期望的是独立同分布的数据，但这并不是现实世界运作的方式。事件随时间发生，并且通常会相应地排列。因此，我们希望在未来的工作中以生物上合理的方式训练CTM。特别是在训练时按顺序采样序列数据（例如视频、文本），是一个有前景的研究方向。

语言模型。我们尚未将CTM应用于语言模型任务，但鉴于它使用注意力机制，将其适应于处理和思考文本是直截了当的。此外，由于它可以构建世界模型并导航该模型（参见第4节），CTM有可能无需位置编码，而是构建一个基于上下文的“世界模型”，反映它所观察到的内容。我们鼓励读者探索将CTM应用于预训练语言模型的可能性。对于未来的工作，我们将构建并训练针对文本数据的定制CTM，以了解其在该领域的功能。

12.4. 什么失败了，我们是如何走到这里的?

我们相信有必要解释我们最初尝试作为CTM核心表示的内容（与同步相对）。激活的潜在空间 \(z_t \) 是一个显而易见的选择。然而，我们发现 \(Z \) 的动态和复杂性质意味着我们需要执行一些平滑操作（例如，累积一些“持有者”潜在变量或logits）。此外，由于这种表示与 \(v5 \) 强烈耦合，我们发现CTM会在损失函数应用时学习何时生成输出（即，何时应用损失函数），而不是依赖于内部自我组织作为主要驱动力。

通过将神经元时间重新引入系统，我们引入了这一挑战；所幸的是，同步性，作为一种与时间无关的特性，证明是一个优雅的解决方案，可以克服这些挑战。可学习的衰减时间依赖性还提供了一种简洁的解决方案，使得CTM可以根据短期神经行为来学习与其环境互动，如果这更受欢迎的话。

13. 结论

The Continuous Thought Machine (CTM) 代表了一种在计算效率与生物可行性之间取得新颖步骤的人工智能方法。通过超越传统的点激活函数，转向私有的神经元级模型，CTM 培育了更为丰富的神经元动力学。至关重要的是，它利用神经同步作为一种强大且从根本上全新的表示方式——这与自神经网络早期以来普遍存在的激活向量截然不同。直接将神经元动力学作为一等表示公民使用，使CTM 能够表现出与当代模型质上不同的行为。

Our research demonstrates the tangible benefits of this approach. The CTM can dynamically build representations over time for tasks like image classification, form rich internal maps to attend to specific input data without positional embeddings, and naturally exhibit adaptive computation. Furthermore, it learns to synchronize neural dynamics to store and retrieve memories beyond its immediate activation history. This internal processing also lends itself to greater interpretability, as seen in its methodical solving of mazes and parity tasks.

Remarkably, the core CTM architecture remained largely consistent across a diverse range of challenging tasks, requiring only input/output module adjustments. This versatility and trainability were particularly evident in complex scenarios like maze navigation. The CTM succeeded with minimal tuning, where a traditional model like the LSTMs still struggled even after significant tuning efforts.

This work underscores a vital, yet often underexplored, synergy between neuroscience and machine learning. While modern AI is ostensibly brain-inspired, the two fields often operate in surprising isolation. The CTM serves as a testament to the power of drawing inspiration from biological principles. By starting with such inspiration and iteratively following the emergent, interesting behaviors, we developed a model with unexpected capabilities, such as its surprisingly strong calibration in classification tasks, a feature that was not explicitly designed for.

It is crucial to note that our approach advocates for borrowing concepts from biology rather than insisting on strict, literal plausibility; real neurons may not access their activation history as modeled in the CTM, yet emergent phenomena like traveling waves still manifest. This nuanced balance between practicality and biological inspiration opens a landscape of new research directions, which may hold the key to unlocking capabilities currently missing in AI, potentially leading to systems that exhibit more human-like intelligence and address its current limitations.

When we initially asked, “why do this research?”, we hoped the journey of the CTM would provide compelling answers. By embracing light biological inspiration and pursuing the novel behaviors observed, we have arrived at a model with emergent capabilities that exceeded our initial designs. We are committed to continuing this exploration, borrowing further concepts to discover what new and exciting behaviors will emerge, pushing the boundaries of what AI can achieve.

Limitations

The main limitation of the CTM is that it requires sequential processing that cannot be parallelized, meaning that training times are longer than a standard feed-forward model, particularly when considering that the current state of modern AI models are well suited to the hardware and software that has developed in parallel (Hooker, 2021).

The additional parameter cost associated with neuron-level models can also be considered a limitation. Whether the benefits outweigh the costs remains to be proven, but we believe their utility can be high.

我们的研究展示了这种方法的实际益处。CTM可以动态地随时间构建表示，用于图像分类等任务，形成丰富的内部地图以关注特定输入数据而无需位置嵌入，并且自然地表现出适应性计算。此外，它学会同步神经动力学以存储和检索超出其即时激活历史的记忆。这种内部处理也使其更具可解释性，如在迷宫和奇偶性任务中的方法解题中所见。

令人惊讶的是，核心CTM架构在各种挑战性任务中保持了很大程度的一致性，只需要调整输入/输出模块。这种灵活性和可训练性在复杂的场景如迷宫导航中尤为明显。CTM在微调后就能成功，而传统的模型如LSTMs即使经过大量调优努力仍然难以应对。

这项工作强调了神经科学和机器学习之间一种至关重要的、但往往被忽视的协同作用。尽管现代AI表面上是受大脑启发的，但这两个领域往往以令人惊讶的方式彼此孤立。CTM 证明了从生物原理中汲取灵感的力量。通过从这种灵感开始，并迭代地跟随出现的有趣行为，我们开发了一个具有意想不到能力的模型，例如其在分类任务中令人惊讶的强校准能力，这一特性并不是明确设计出来的。

需要注意的是，我们的方法提倡借鉴生物学概念而非坚持严格的、字面意义上的可行性；实际的神经元可能不会像CTM中那样访问其激活历史，但像行波这样的涌现现象仍然会出现。这种在实用性和生物学启发之间的微妙平衡开辟了新的研究方向，这些方向可能包含了解锁当前AI所缺乏的能力的关键，从而可能导致表现出更多人类智能特征的系统，并解决其当前的局限性。

当最初我们问“为什么要进行这项研究”时，我们希望CTM的旅程能提供令人信服的答案。通过拥抱轻量级的生物启发并追求观察到的新奇行为，我们到达了一个具有超出初始设计 emergent 能力的模型。我们致力于继续这一探索，借鉴更多的概念以发现新的、令人兴奋的行为将如何涌现，推动AI所能实现的边界。

限制
CTM的主要限制是它需要顺序处理，无法并行化，这意味着训练时间比标准前馈模型更长，尤其是在考虑到当前现代AI模型非常适合与并行发展的硬件和软件（Hooker, 2021）的情况下。

与神经元级模型相关的额外参数成本也可以被视为一个局限性。尽管效益是否超过成本仍有待证明，但我们认为它们的实用性可以很高。

References

Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In *International conference on machine learning*, pages 242–252. PMLR, 2019.

Cristina M Atance and Daniela K O'Neill. Episodic future thinking. *Trends in cognitive sciences*, 5(12): 533–539, 2001.

Andrea Banino, Jan Balaguer, and Charles Blundell. Pondernet: Learning to ponder. *arXiv preprint arXiv:2107.05407*, 2021.

Arpit Bansal, Avi Schwarzschild, Eitan Borgnia, Zeyad Emam, Furong Huang, Micah Goldblum, and Tom Goldstein. End-to-end algorithm synthesis with recurrent networks: Extrapolation without overthinking. *Advances in Neural Information Processing Systems*, 35:20232–20242, 2022.

Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846, 1983. doi: 10.1109/TSMC.1983.6313077.

Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. Adaptive neural networks for efficient inference. In *International Conference on Machine Learning*, pages 527–536. PMLR, 2017.

Natalia Caporale and Yang Dan. Spike timing–dependent plasticity: a hebbian learning rule. *Annu. Rev. Neurosci.*, 31(1):25–46, 2008.

Peter Cariani and Janet M Baker. Time is of the essence: neural codes, synchronies, oscillations, architectures. *Frontiers in Computational Neuroscience*, 16:898829, 2022.

Makram Chahine, Ramin Hasani, Patrick Kao, Aaron Ray, Ryan Shubert, Mathias Lechner, Alexander Amini, and Daniela Rus. Robust flight navigation out of distribution with liquid neural networks. *Science Robotics*, 8(77):eadc8892, 2023.

Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo de Lazcano, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and Jordan Terry. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. *CoRR*, abs/2306.13831, 2023.

François Chollet. On the measure of intelligence. *arXiv preprint arXiv:1911.01547*, 2019.

Francois Chollet, Mike Knoop, Gregory Kamradt, and Bryan Landers. Arc prize 2024: Technical report. *arXiv preprint arXiv:2412.04604*, 2024.

Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *International conference on machine learning*, pages 933–941. PMLR, 2017.

Rahul Dey and Fathi M Salem. Gate-variants of gated recurrent unit (gru) neural networks. In *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*, pages 1597–1600. IEEE, 2017.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *arXiv preprint arXiv:2502.05171*, 2025.

References Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. 通过过参数化实现深度学习的收敛理论. 在 *International conference on machine learning*，第242-252页. PMLR, 2019. Cristina M Atance和Daniela K O’Neill. 情景未来的思考. *Trends in cognitive sciences*，5(12): 533–539, 2001. Andrea Banino, Jan Balaguer, and Charles Blundell. Pondernet：学习思考. *arXiv preprint arXiv:2107.05407*，2021. Arpit Bansal, Avi Schwarzschild, Eitan Borgnia, Zeyad Emam, Furong Huang, Micah Goldblum, and Tom Goldstein. 基于递归网络的端到端算法合成：无需过度思考的外推. *Advances in Neural Information Processing Systems*，35:20232–20242, 2022. A

ndrew G. Barto, Richard S. Sutton, 和 Charles W. Anderson. 具有类似神经元的自适应元素，可以解决复杂的控制学习问题。*IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846, 1983. doi: 10.1109/TSMC.1983.6313077.

Tolga Bolukbasi, Joseph Wang, Ofer Dekel, 和 Venkatesh Saligrama. 适应性神经网络以实现高效推理。在 *International Conference on Machine Learning* 中，第 527–536 页。PMLR，2017。Natalia Caporale 和 Yang Dan. 神经元放电时间依赖可塑性：一个希伯学习规则。*Annu. Rev. Neurosci.*，第 31 卷第 1 期：25–46，2008。Peter Cariani 和 Janet M Baker. 时间至关重要：神经编码、同步、振荡、架构。*Frontiers in Computational Neuroscience*，第 16 期：898829，2022。M

阿克拉姆·查海因,拉敏·哈萨尼,帕特里克·高,亚伦·雷,瑞安·舒伯特,马蒂亚斯·雷尼希,亚历山大·阿米尼,和丹妮拉·鲁斯. 使用液态神经网络的离分布稳健飞行导航.*Science Robotics*, 8(77):eadc8892, 2023.

Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo de Lazcano, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and Jordan Terry. Minigrid & miniworld: 模块化且可定制的强化学习环境，用于目标导向任务。*CoRR*, abs/2306.13831, 2023.

François Chollet. 智能的度量. *arXiv preprint arXiv:1911.01547*, 2019. F

rancois Chollet, Mike Knoop, Gregory Kamradt, 和 Bryan Landers. Arc prize 2024: 技术报告。*arXiv preprint arXiv:2412.04604*, 2024.

Yann N Dauphin, Angela Fan, Michael Auli, 和 David Grangier. 使用门控卷积网络的语言模型。在 *International conference on machine learning*，第 933–941 页。PMLR，2017 年。

Rahul Dey 和 Fathi M Salem. 门型循环单元（GRU）神经网络的门变体。在 *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*，页码 1597–1600。IEEE，2017. J

Jonathan Frankle 和 Michael Carbin. 彩票票假设：寻找稀疏可训练神经网络。*arXiv preprint arXiv:1803.03635*，2018年。

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. 用潜推理扩展测试时计算：一种递归深度方法。*arXiv preprint arXiv:2502.05171*, 2025.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*. MIT press Cambridge, 2016.

James Gornet and Matt Thomson. Automated construction of cognitive maps with visual predictive coding. *Nature Machine Intelligence*, 6(7):820–833, 2024.

Anirudh Goyal, Alex Lamb, Jordan Hoffmann, Shagun Sodhani, Sergey Levine, Yoshua Bengio, and Bernhard Schölkopf. Recurrent independent mechanisms. *arXiv preprint arXiv:1909.10893*, 2019.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376, 2006.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International conference on machine learning*, pages 1321–1330. PMLR, 2017.

David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

Ramin Hasani, Mathias Lechner, Alexander Amini, Daniela Rus, and Radu Grosu. Liquid time-constant networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 7657–7666, 2021.

Matthew J Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *AAAI fall symposia*, volume 45, page 141, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology press, 2005.

Tien Ho-Phuoc. Cifar10 to compare visual recognition performance between deep neural networks and humans. *arXiv preprint arXiv:1811.07270*, 2018.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Patrick Hohenecker and Thomas Lukasiewicz. Ontology reasoning with deep neural networks. *Journal of Artificial Intelligence Research*, 68:503–540, 2020.

Sara Hooker. The hardware lottery. *Communications of the ACM*, 64(12):58–65, 2021.

Michael Igorevich Ivanitskiy, Rusheb Shah, Alex F. Spies, Tilman Räuker, Dan Valentine, Can Rager, Lucia Quirke, Chris Mathwin, Guillaume Corlouer, Cecilia Diniz Behn, and Samy Wu Fung. A configurable library for generating and manipulating maze datasets, 2023. URL http://arxiv.org/abs/2309.10498.

Mozes Jacobs, Roberto C Budzinski, Lyle Muller, Demba Ba, and T Anderson Keller. Traveling waves integrate spatial information into spectral representations. *arXiv preprint arXiv:2502.06034*, 2025.

Herbert Jaeger. Echo state network. *scholarpedia*, 2(9):2330, 2007.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, 和 Yoshua Bengio. *Deep learning*. MIT press Cambrid ge, 2016.

James Gornet 和 Matt Thomson. 基于视觉预测编码的自动认知图构建. *Nature Machine Intelligence*, 6(7):820–833, 2024.

Anirudh Goyal, Alex Lamb, Jordan Hoffmann, Shagun Sodhani, Sergey Levine, Yoshua Bengio, 和 Bernha rd Schölkopf. 循环独立机制。*arXiv preprint arXiv:1909.10893*, 2019。

Alex Graves. 适应性计算时间用于循环神经网络。*arXiv preprint arXiv:1603.08983*, 2016. Alex Grav es, Santiago Fernández, Faustino Gomez, 和 Jürgen Schmidhuber. 连接主义时间分类：使用循环神经网 络标注未分割序列数据。在 *Proceedings of the 23rd international conference on Machine learning*, 页 码 369–376, 2006.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. 关于现代神经网络的校准. 在 *International conference on machine learning*，第 1321–1330 页. PMLR, 2017.

David Ha 和 Jürgen Schmidhuber. 世界模型。*arXiv preprint arXiv:1803.10122*, 2018.

Ramin Hasani, Mathias Lechner, Alexander Amini, Daniela Rus, 和 Radu Grosu. 液体时间常数网络. 在 *Proceedings of the AAAI Conference on Artificial Intelligence*，卷 35，页码 7657–7666，2021。

Matthew J Hausknecht 和 Peter Stone. 深度递归Q学习在部分可观测MDPs中的应用. 在 *AAAI fall symposia*，卷 45，第 141 页，2015年.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 深度残差学习在图像识别中的应用. 在 *Proceedings of the IEEE conference on computer vision and pattern recognition*, 页码 770–778, 2016.

DDonald Olding Hebb. *The organization of behavior: A neuropsychological theory*. 心理学出版社, 200 5年。

Tien Ho-Phuoc. Cifar10 以比较深度神经网络和人类的视觉识别性能。 *arXiv preprint arXiv:1811.07270*, 2018。

Sepp Hochreiter 和 Jürgen Schmidhuber. 长短时记忆。*Neural computation*, 9(8): 1735–1780, 1997.

Patrick Hohenecker 和 Thomas Lukasiewicz. 使用深度神经网络进行本体推理. *Journal of Artificial Intelligence Research*, 68:503–540, 2020.

Sara Hooker. 硬件彩票。*Communications of the ACM*, 64(12):58–65, 2021。

Michael Igorevich Ivanitskiy, Rusheb Shah, Alex F. Spies, Tilman Räuker, Dan Valentine, Can Rager, Lucia Quirke, Chris Mathwin, Guillaume Corlouer, Cecilia Diniz Behn, 和 Samy Wu Fung. 可配置的生成和操作 迷宫数据集的库，2023。URL `http://arxiv.org/abs/2309.10498`。

莫泽斯·雅各布斯,罗伯托·C·布兹金斯基,莱尔·穆勒,达姆巴·巴,和泰勒·安德森·凯勒. 行波将空间信息整 合到频谱表示中。*arXiv preprint arXiv:2502.06034*, 2025.

Herbert Jaeger. Echo state network. *scholarpedia*, 2(9):2330, 2007.

Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In *International conference on machine learning*, pages 4651–4664. PMLR, 2021.

Louis Kirsch and Jürgen Schmidhuber. Meta learning backpropagation and improving it. *Advances in Neural Information Processing Systems*, 34:14122–14134, 2021.

Louis Kirsch, Sebastian Flennerhag, Hado Van Hasselt, Abram Friesen, Junhyuk Oh, and Yutian Chen. Introducing symmetries to black box meta reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 7202–7210, 2022.

Dhireesha Kudithipudi, Mario Aguilar-Simon, Jonathan Babb, Maxim Bazhenov, Douglas Blackiston, Josh Bongard, Andrew P Brna, Suraj Chakravarthi Raja, Nick Cheney, Jeff Clune, et al. Biological underpinnings for lifelong learning machines. *Nature Machine Intelligence*, 4(3):196–210, 2022.

Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Building machines that learn and think like people. *Behavioral and brain sciences*, 40:e253, 2017.

Yann LeCun. A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27. *Open Review*, 62(1):1–62, 2022.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Wolfgang Maass. On the relevance of time in neural computation and learning. *Theoretical Computer Science*, 261(1):157–178, 2001.

Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. *Advances in neural information processing systems*, 31, 2018.

Gary Marcus. Deep learning: A critical appraisal. *arXiv preprint arXiv:1801.00631*, 2018.

Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

Larry Medsker and Lakhmi C Jain. *Recurrent neural networks: design and applications*. CRC press, 1999.

Takeru Miyato, Sindy Löwe, Andreas Geiger, and Max Welling. Artificial kuramoto oscillatory neurons. *arXiv preprint arXiv:2410.13821*, 2024.

Lyle Muller, Frédéric Chavane, John Reynolds, and Terrence J Sejnowski. Cortical travelling waves: mechanisms and computational principles. *Nature Reviews Neuroscience*, 19(5):255–268, 2018.

Elias Najarro and Sebastian Risi. Meta-learning through hebbian plasticity in random networks. *Advances in Neural Information Processing Systems*, 33:20719–20731, 2020.

Joachim Pedersen, Erwan Plantec, Eleni Nisioti, Milton Montero, and Sebastian Risi. Structurally flexible neural networks: Evolving the building blocks for general agents. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1119–1127, 2024.

Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, 和 Joao Carreira. Perceiver: 通用迭代注意力感知. 在 *International conference on machine learning* 中，第 4651–4664 页。PMLR, 2021。

Louis Kirsch 和 Jürgen Schmidhuber. 元学习反向传播并改进它。*Advances in Neural Information Processing Systems*, 34:14122–14134, 2021.

Louis Kirsch, Sebastian Flennerhag, Hado Van Hasselt, Abram Friesen, Junhyuk Oh, 和 Yutian Chen. 将对称性引入黑盒元强化学习。在 *Proceedings of the AAAI Conference on Artificial Intelligence*，卷 36，页码 7202–7210，2022。

D雇员哈里埃沙·库迪蒂普迪,马里奥·阿吉拉尔-西蒙,乔纳森·巴布,马克斯·巴济诺夫,道格拉斯·布莱克斯顿,约什·邦加德,安德鲁·P·布rna,苏拉杰·查克拉瓦里拉贾,尼克·切尼,杰夫·克鲁内,等. 终身学习机器的生物基础. *Nature Machine Intelligence*, 4(3):196–210, 2022.

Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, 和 Samuel J Gershman. 建立像人一样学习和思考的机器。*Behavioral and brain sciences*, 40:e253, 2017.

Yann LeCun. 通向自主机器智能的道路版本 0.9. 2, 2022-06-27. *Open Review*, 62(1):1–62, 2022.

Yann LeCun, Léon Bottou, Yoshua Bengio, 和 Patrick Haffner. 基于梯度的学习在文档识别中的应用. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Yann LeCun, Yoshua Bengio, 和 Geoffrey Hinton. 深度学习. *nature*, 521(7553):436–444, 2015.

Ilya Loshchilov和Frank Hutter. 分解权重衰减正则化。*arXiv preprint arXiv:1711.05101*，2017。

Wolfgang Maass. 关于时间在神经计算和学习中的相关性。*Theoretical Computer Science*, 261(1):157–178, 2001。

Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, 和 Luc De Raedt. Deepproblog: 神经概率逻辑编程. *Advances in neural information processing systems*, 31, 2018.

Gary Marcus. 深度学习：批判性评价。*arXiv preprint arXiv:1801.00631*, 2018.

Leland McInnes, John Healy, 和 James Melville. Umap：均匀流形逼近和投影在降维中的应用. *arXiv preprint arXiv:1802.03426*, 2018.

Larry Medsker 和 Lakhmi C Jain. *Recurrent neural networks: design and applications*. CRC press, 1999.

Takeru Miyato, Sindy Löwe, Andreas Geiger, 和 Max Welling. 人工库拉莫振荡神经元。*arXiv preprint arXiv:2410.13821*, 2024.

Lyle Muller, Frédéric Chavane, John Reynolds, 和 Terrence J Sejnowski. 枕叶 traveling waves：机制与计算原理。*Nature Reviews Neuroscience*, 19(5):255–268, 2018.

Elias Najarro 和 Sebastian Risi. 通过海　安可塑性在随机网络中的元学习。*Advances in Neural Information Processing Systems*, 33:20719–20731, 2020.

乔achim Pedersen, Erwan Plantec, Eleni Nisioti, Milton Montero, 和 Sebastian Risi. 结构上灵活的神经网络：为通用代理演化构建模块。在 *Proceedings of the Genetic and Evolutionary Computation Conference*，页码 1119–1127，2024。

Joshua C Peterson, Ruairidh M Battleday, Thomas L Griffiths, and Olga Russakovsky. Human uncertainty makes classification more robust. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9617–9626, 2019.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, et al. Humanity's last exam, 2025. URL https://arxiv.org/abs/2501.14249.

Jing Ren and Feng Xia. Brain-inspired artificial intelligence: A comprehensive review. *arXiv preprint arXiv:2408.14811*, 2024.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pages 234–241. Springer, 2015.

Imanol Schlag and Jürgen Schmidhuber. Augmenting classic algorithms with neural components for strong generalisation on ambiguous and high-dimensional data. In *Advances in Programming Languages and Neurosymbolic Systems Workshop*, 2021.

Samuel Schmidgall, Rojin Ziaei, Jascha Achterberg, Louis Kirsch, S Hajiseyedrazi, and Jason Eshraghian. Brain-inspired learning in artificial neural networks: a review. *APL Machine Learning*, 2(2), 2024.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Avi Schwarzschild, Eitan Borgnia, Arjun Gupta, Furong Huang, Uzi Vishkin, Micah Goldblum, and Tom Goldstein. Can you learn an algorithm? generalizing from easy to hard problems with recurrent networks. *Advances in Neural Information Processing Systems*, 34:6695–6706, 2021.

Matei-Ioan Stan and Oliver Rhodes. Learning long sequences in spiking neural networks. *Scientific Reports*, 14(1):21957, 2024.

Richard S Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in neural information processing systems*, 8, 1995.

Shawn Tan, Yikang Shen, Zhenfang Chen, Aaron Courville, and Chuang Gan. Sparse universal transformer. *arXiv preprint arXiv:2310.07096*, 2023.

Neil C Thompson, Kristjan Greenewald, Keeheon Lee, Gabriel F Manso, et al. The computational limits of deep learning. *arXiv preprint arXiv:2007.05558*, 10, 2020.

Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A standard interface for reinforcement learning environments, 2024. URL https://arxiv.org/abs/2407.17032.

Peter Uhlhaas, Gordon Pipa, Bruss Lima, Lucia Melloni, Sergio Neuenschwander, Danko Nikolić, and Wolf Singer. Neural synchrony in cortical networks: history, concept and current status. *Frontiers in integrative neuroscience*, 3:543, 2009.

Paul E Utgoff. *Machine learning of inductive bias*, volume 15. Springer Science & Business Media, 2012.

Joshua C Peterson, Ruairidh M Battleday, Thomas L Griffiths, 和 Olga Russakovsky. 人类不确定性使分类更加稳健。在 *Proceedings of the IEEE/CVF international conference on computer vision*，第 9617–9626 页，2019 年。

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, 等. 人类的最后一考，2025. URL https://arxiv.org/abs/2501.14249.

Jing Ren和Feng Xia. 类脑人工智能：全面综述. *arXiv preprint arXiv:2408.14811*, 2024.

Olaf Ronneberger, Philipp Fischer, 和 Thomas Brox. U-net: 卷积网络在生物医学图像分割中的应用. 在 *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18* 中，第 234–241 页. 春花ter，2015年.

我anol Schlag 和 Jürgen Schmidhuber. 通过加入神经组件增强经典算法，以在模糊和高维数据上实现强大的泛化能力。在 *Advances in Programming Languages and Neurosymbolic Systems Workshop*，2021。

Samuel Schmidgall, Rojin Ziaei, Jascha Achterberg, Louis Kirsch, S Hajiseyedrazi, 和 Jason Eshraghian. 基于大脑的学习在人工神经网络中的应用：一篇综述。*APL Machine Learning*, 2(2), 2024。

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, 和 Oleg Klimov. 逼近策略优化算法。*arXiv preprint arXiv:1707.06347*, 2017.

Avi Schwarzschild, Eitan Borgnia, Arjun Gupta, Furong Huang, Uzi Vishkin, Micah Goldblum, 和 Tom Goldstein. 你可以从简单的到困难的问题学习一个算法吗？使用循环网络进行泛化。*Advances in Neural Information Processing Systems*, 34:6695–6706, 2021.

Matei-Ioan Stan 和 Oliver Rhodes. 在脉冲神经网络中学习长序列。*Scientific Reports*, 14(1):21957, 2024。

Richard S Sutton. 强化学习中的泛化：使用稀疏粗编码的成功案例。*Advances in neural information processing systems*, 8, 1995。

Shawn Tan, Yikang Shen, Zhenfang Chen, Aaron Courville, 和 Chuang Gan. 稀疏通用变换器。*arXiv preprint arXiv:2310.07096*, 2023.

Neil C Thompson, Kristjan Greenewald, Keeheon Lee, Gabriel F Manso, 等. 深度学习的计算极限。*arXiv preprint arXiv:2007.05558*, 10, 2020.

Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, 和 Omar G. Younis. Gymnasium: 一种强化学习环境的标准接口，2024. URL https://arxiv.org/abs/2407.17032.

Peter Uhlhaas, Gordon Pipa, Bruss Lima, Lucia Melloni, Sergio Neuenschwander, Danko Nikoli , 和 Wolf Singer. 枕叶网络中的神经同步：历史、概念和当前状态。*Frontiers in integrative neuroscience*, 3:543, 2009。

Paul E Utgoff. *Machine learning of inductive bias*, 体积 15. 春花科学与商业媒体, 2012年。

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Vasilis Vryniotis and Matthieu Cord. How to Train State-Of-The-Art Models Using TorchVision's Latest Primitives. PyTorch Blog, dec 2021. URL https://pytorch.org/blog/how-to-train-state-of-the-art-models-using-torchvision-latest-primitives/. Last edited on 2021-12-22.

Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057. PMLR, 2015.

Fuzhao Xue, Valerii Likhosherstov, Anurag Arnab, Neil Houlsby, Mostafa Dehghani, and Yang You. Adaptive computation with elastic input sequence. In *International Conference on Machine Learning*, pages 38971–38988. PMLR, 2023.

Liu Yang, Kangwook Lee, Robert Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. *arXiv preprint arXiv:2311.12424*, 2023.

Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D Goodman. Quiet-star: Language models can teach themselves to think before speaking. *arXiv preprint arXiv:2403.09629*, 2024.

Tao Zhang, Jia-Shu Pan, Ruiqi Feng, and Tailin Wu. T-scend: Test-time scalable mcts-enhanced diffusion model. *arXiv preprint arXiv:2502.01989*, 2025.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, ukasz Kaiser, and Illia Polosukhin. 注意力是你需要的一切。*Advances in neural information processing systems*, 30, 2017.

Vasilis Vryniotis 和 Matthieu Cord. 如何使用 TorchVision 的最新原语训练最先进的模型. PyTorch 博客, 2021 年 12 月. URL `https://pytorch.org/blog/how-to-train-state-of-the-art-models-using-torchvision-latest-primitives/`. 最后编辑于 2021-12-22. Liyuan Wang, Xingxing Zhang, Hang Su, 和 Jun Zhu. 持续学习的全面综述：理论、方法和应用. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024. Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, 等. 大型语言模型的涌现能力. *arXiv preprint arXiv:2206.07682*, 2022. Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, 和 Yoshua Bengio. Show, attend and tell: 带有视觉注意力的神经图像字幕生成. 在 *International conference on machine learning* 中, 页码 2048–2057. PMLR, 2015. Fuzhao Xue, Valerii Likhosherstov, Anurag Arnab, Neil Houlsby, Mostafa Dehghani, 和 Yang You. 适应性计算与弹性输入序列. 在 *International Conference on Machine Learning* 中, 页码 38971–38988. PMLR, 2023. Liu Yang, Kangwook Lee, Robert Nowak, 和 Dimitris Papailiopoulos. 闭环变压器在学习学习算法方面更有效. *arXiv preprint arXiv:2311.12424*, 2023.

Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, 和 Noah D Goodman. Quiet-star: 语言模型可以自学在说话之前思考。*arXiv preprint arXiv:2403.09629*, 2024。

T张傲, 半嘉舒, 冯瑞琪, 和 吴 Tailin. T-scend: 测试时可扩展的MCTS增强扩散模型. *arXiv preprint arXiv:2502.01989*, 2025.

A. Glossary

Term	Description
Internal tick	One step of internal computation.
Memory length	Length of rolling history of pre-activations, updated in a rolling FIFO fashion
Synapse model	Recurrent model that takes $\mathbf{z}^t$ and $\mathbf{o}^t$ as input to preduct pre-activations, $\mathbf{a}^t$.
Pre-activations	Output of recurrent synapse model, input to NLMs.
Post-activations	Output of NLMs, neuron states at time t.
(Pre/Post) activation history	Sequentially ordered history of activations over time.
Neuron-Level Model (NLM)	Per-neuron MLP over pre-activation history.
Synchronization	Dot product of post-activation histories.
Self-pair	Diagonal synchronization matrix entries (i, i).
Action synchronization	Synchronization representation for attention queries.
Output synchronization	Synchronization representation for predictions.
Decay r_{ij}	Learnable time decay for synchronization (action or output).
Feature extractor	Task-specific input encoder (e.g., ResNet).
Attention output	Output after cross-attention using queries, $\mathbf{q}^t$, computed from action synchronization, and keys/values from data.

Table 1 | Glossary of terms.

Symbol	Meaning
T	Number of internal ticks
M	Memory length
d_{model}	Dimensionality of latent state in the CTM
d_{input}	Dimensionality of attention output
d_{hidden}	Hidden size in each neuron's private MLP (NLM)
k	Depth of the synapse MLP or U-Net
p_{dropout}	Dropout probability in the synapse model
n_{heads}	Number of heads in multi-head attention
J_{action}	Number of neurons used for action synchronization
J_{out}	Number of neurons used for output synchronization
D_{action}	Dimensionality of action synchronization vector
D_{out}	Dimensionality of output synchronization vector
n_{self}	Number of self-pairs (i, i) used in synchronization sampling
r_{ij}	Learnable decay parameter for synchronization between neuron i and j
$\mathbf{S}^t$	Full synchronization matrix at internal tick t
$\mathbf{q}^t$	Query vector projected from action synchronization
$\mathbf{y}^t$	Output vector (e.g., logits) projected from output synchronization

Table 2 | Glossary of symbols.

B. Method details

B.1. Synapse models

Figure 29 show the synapse model which is the recurrent structure that shares information across neurons in the CTM. It is implemented by choosing a depth of k (always even), where each subsequent

A. 术语表

Term	Description
Internal tick	One step of internal computation.
Memory length	Length of rolling history of pre-activations, updated in a rolling FIFO fashion
Synapse model	Recurrent model that takes $\mathbf{z}^t$ and $\mathbf{o}^t$ as input to preduct pre-activations, $\mathbf{a}^t$.
Pre-activations	Output of recurrent synapse model, input to NLMs.
Post-activations	Output of NLMs, neuron states at time t.
(Pre/Post) activation history	Sequentially ordered history of activations over time.
Neuron-Level Model (NLM)	Per-neuron MLP over pre-activation history.
Synchronization	Dot product of post-activation histories.
Self-pair	Diagonal synchronization matrix entries (i, i).
Action synchronization	Synchronization representation for attention queries.
Output synchronization	Synchronization representation for predictions.
Decay r_{ij}	Learnable time decay for synchronization (action or output).
Feature extractor	Task-specific input encoder (e.g., ResNet).
Attention output	Output after cross-attention using queries, $\mathbf{q}^t$, computed from action synchronization, and keys/values from data.

表1 | 术语表。

Symbol	Meaning
T	Number of internal ticks
M	Memory length
d_{model}	Dimensionality of latent state in the CTM
d_{input}	Dimensionality of attention output
d_{hidden}	Hidden size in each neuron's private MLP (NLM)
k	Depth of the synapse MLP or U-Net
p_{dropout}	Dropout probability in the synapse model
n_{heads}	Number of heads in multi-head attention
J_{action}	Number of neurons used for action synchronization
J_{out}	Number of neurons used for output synchronization
D_{action}	Dimensionality of action synchronization vector
D_{out}	Dimensionality of output synchronization vector
n_{self}	Number of self-pairs (i, i) used in synchronization sampling
r_{ij}	Learnable decay parameter for synchronization between neuron i and j
$\mathbf{S}^t$	Full synchronization matrix at internal tick t
$\mathbf{q}^t$	Query vector projected from action synchronization
$\mathbf{y}^t$	Output vector (e.g., logits) projected from output synchronization

表 2 | 符号 glossary.

B. 方法细节

B.1. 突触模型

F图29显示了该突触模型，这是一种在不同位置共享信息的递归结构
nCTM中的神经元。它通过选择一个深度为k (总是偶数)来实现，其中每个后续

layer width is chosen to linearly reduce the dimensionality until a width of 16 is reached, and then increase thereafter, using skip connections to retain information. The synapse model also takes $\mathbf{o}^t$ (the output of attention) as input.

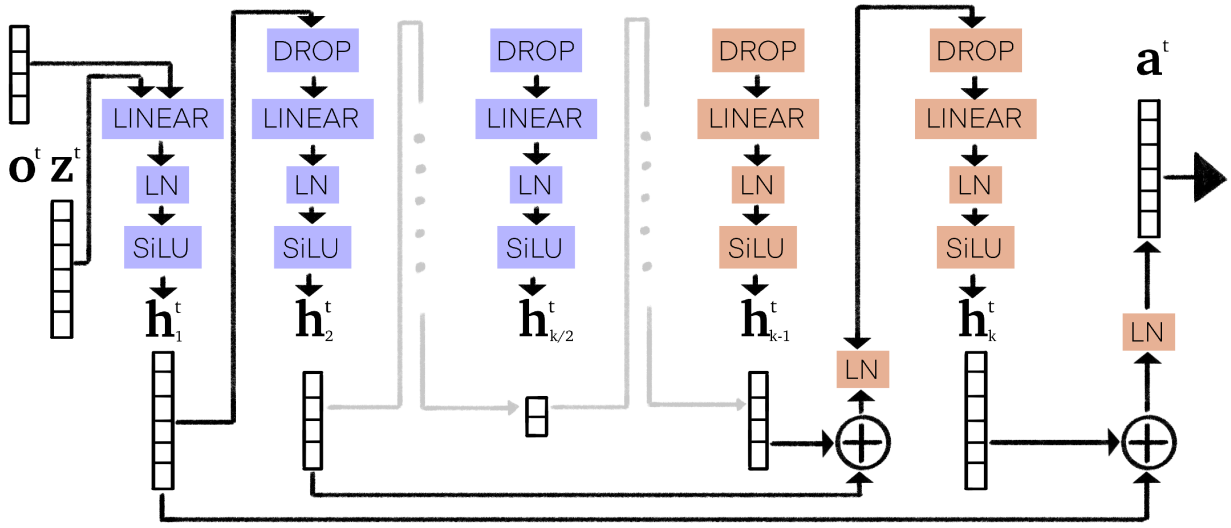


Figure 29 | Overview of UNET style 'synapse' recurrent models. $\mathbf{z}^t$ are the post-activations from the previous step, $\mathbf{o}^t$ are the attention outputs from observing data, and the synapse model yields $\mathbf{a}^t$ pre-activations for the NLMs to process. The UNET structure is set up so that the innermost bottleneck layer is 16 units wide, with linear scaling for each layer in between. Skip connections with layer-norm implement the classic UNET structure to maintain information. Layers that produce lower dimensionality are shown in blue, while layers that produce higher dimensionality are shown in orange.

B.2. Sampling synchronization neurons

The CTM operates using recurrence on a latent representation, $\mathbf{z}$, that is D-dimensional. $\mathbf{z}$ unfolds over time and the synchronization between *some chosen neurons* form the new kind of representation that the CTM enables.

There are $\frac{D\times(D+1)}{2}$ unique pairs of neurons, making for a substantially **larger set of neuron synchronization pairs** than there are neurons themselves. This motivates the need for a selection process. Over the development of the CTM we came up with three approaches to selecting neurons:

1. **Dense pairing**: in this setup we select J neurons and compute synchronization for every possible (i, j) pair of the J neurons. For $\mathbf{S}^t_{\text{out}}$ we choose J_{out} neurons and for $\mathbf{S}^t_{\text{action}}$ we choose non-overlapping J_{action} neurons. Selecting J_{out} neurons for dense pairing results in an output synchronization representation of $D_{\text{out}} = \frac{J_{\text{out}}\times(J_{\text{out}}+1)}{2}$, and similarly for the action representation. This approach essentially creates a strong bottleneck where all gradients must flow through the selected neurons, which can be advantageous for some tasks.
2. **Semi-dense pairing**: in this setup we open up the aforementioned bottleneck twofold by selecting two different subsets, J_1 and J_2, such that the left neurons of the synchronization dot product, i, are taken from J_1 and the right neurons, j, are taken from J_2. The same dense computation as before is then applied.
   The bottleneck width in this case is 2× as wide as before. Output and action selections are, once more, not overlapping.
3. **Random pairing**: in this setup we randomly select D_{out} or D_{action} pairs of neurons and compute the synchronization between each pair as opposed to doing so densely between all selected neurons. We also intentionally compute the (i, i) dot products between n_{self} neurons in each case in order to ensure that the CTM could recover a snapshot representation if it wanted to.

层宽选择线性减少维度直到宽度达到16，之后再增加，使用跳连保留信息。神经元模型也将注意力^t(的输出)作为输入。

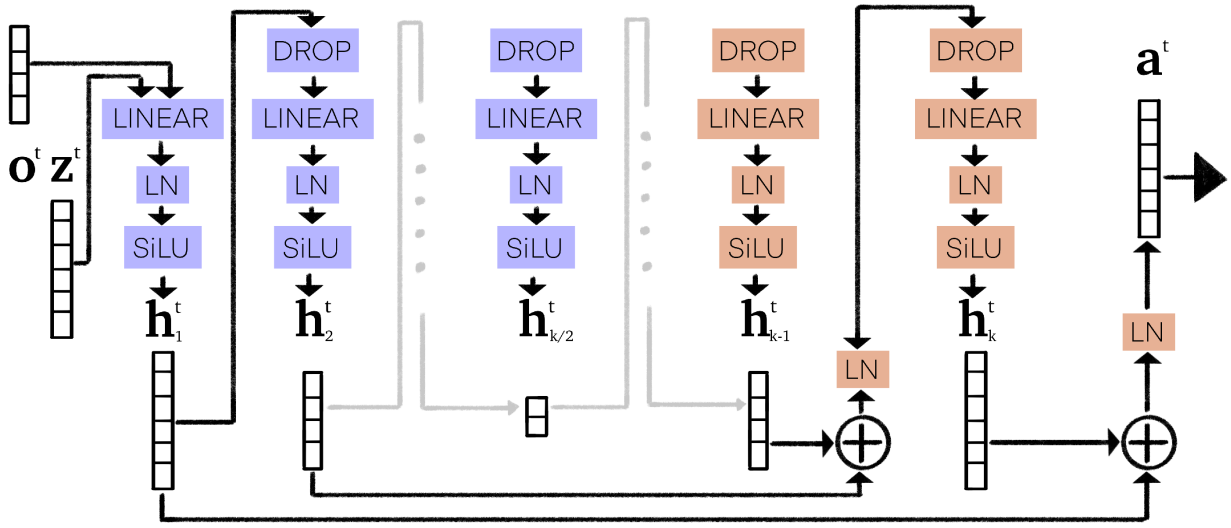


图 29 | UNET风格的‘synapse’循环模型概述。z^t是从上一步的后激活输出，o^t是从观察数据得到的注意力输出，synapse模型生成a^t预激活供NLM处理。UNET结构设置为内部瓶颈层宽度为16个单位，每层之间线性缩放。带有层规范化（LayerNorm）的跳连实现经典的UNET结构以保持信息。生成较低维度的层用蓝色显示，生成较高维度的层用橙色显示。

B.2. 取样同步神经元

T他使用的CTM使用递归在一个潜在表示z上操作，z是D-维的。z展开
o随时间推移和 *some chosen neurons* 的同步形成新的表示形式
tCTM 是否使能。

存在 $\frac{D\times(D+1)}{2}$ 个独特的神经元配对，使得神经元同步配对集比神经元本身要大得多。这促使了选择过程的需求。在CTM的发展过程中，我们提出了三种选择神经元的方法：

1. 密集配对：在这种设置中，我们选择 J 神经元，并计算每对 (i, j) 的 J 神经元的同步。对于 S^t_{out}，我们选择 J_{out} 神经元，而对于 S^t_{action}，我们选择不重叠的 J_{action} 神经元。选择 J_{out} 神经元进行密集配对会产生一个输出同步表示 $D_{\text{out}} = \frac{J_{\text{out}}\times(J_{\text{out}}+1)}{2}$，类似地，对于动作表示也是如此。这种方法本质上创建了一个强瓶颈，其中所有梯度都必须通过选定的神经元流动，这对于某些任务可能是有利的。

2. 半密集配对：在这种设置中，我们通过选择两个不同的子集 J_1 和 J_2 来双重解决上述瓶颈，使得同步点积的左神经元 i 来自 J_1，右神经元 j 来自 J_2。然后应用与之前相同的密集计算。

   在这种情况下，瓶颈宽度是2×与之前相同。输出和行动选择再次不重叠。

3. 随机配对：在这种设置中，我们随机选择 D_{out} 或 D_{action} 对神经元，并计算每对之间的同步性，而不是在所有选定的神经元之间密集地进行计算。我们还在每种情况下故意计算 (i, i) 神经元之间的点积，以确保CTM能够恢复一个快照表示，如果它想要的话。

This opens up the bottleneck much more than before. We allow overlapping selections in this case.

C. ImageNet-1K

This section provides additional details and results for the ImageNet-1K experiments.

C.1. Architecture details

We used a constrained version of the classic ResNet architecture (He et al., 2016) for this task, which we adapted from https://github.com/huyvnphan/PyTorch_CIFAR10. It differs from the standard implementation for ImageNet in that the first convolution is constrained to use a kernel size of 3×3 as opposed to 7×7. We used a ResNet-152 structure and took the output prior to the final average pooling and projection to class logits. We used input images of size 224×224 which yielded 14×14 features for the keys and values used in cross attention.

We used the following hyperparameters:

- $D = 4096$ (the width of $\mathbf{z}^t$ and $\mathbf{a}^t$)
- $k = 16$ (synapse depth, 8 layers down and 8 layers up)
- $d_{\text{input}} = 1024$ (the width of attention output, $\mathbf{o}^t$)
- $n_{\text{heads}} = 16$
- Random pairing for neuron selection (see Appendix B.2)
- $D_{\text{out}} = 8196$ (width of $\mathbf{S}^t_{\text{out}}$ synchronization representation)
- $D_{\text{action}} = 2048$ (width of $\mathbf{S}^t_{\text{action}}$ synchronization representation)
- $n_{\text{self}} = 32$ (for recovering a snapshot representation)
- $T = 50$ (internal ticks)
- $M = 25$ (FIFO rolling memory input to NLMs)
- $d_{\text{hidden}} = 64$ (width of MLPs inside NLMs)
- $p_{\text{dropout}} = 0.2$ (dropout probability for synapse model)
- No positional embedding

We used the following settings for optimization:

- Trained using a batch size of 64 across 8 H100 Nvidia GPUs
- 500000 iterations for training, using a custom sampling such that each minibatch was sampled with possible replacement
- AdamW (Loshchilov and Hutter, 2017)
- A learning rate of 5e-4 with a linear warmup of 10000 iterations and decaying to zero using a cosine annealing learning rate scheduler
- Gradient norm clipping set to a norm of 20
- No weight decay

C.2. Loss function

Listing 5 shows the python code for the image classification loss function used to train the CTM on ImageNet-1K. This accompanies the loss defined in Section 2.5.

C.3. Additional demonstrations

D. 2D Mazes

D.1. Dataset

We used the maze-dataset repository to generate mazes for this work: https://github.com/understanding-search/maze-dataset. We generated mazes of size 19×19, 39×39, and 99×99.

这个瓶颈被打开得比以前更多。在这种情况下，我们允许重叠的选择。

C. ImageNet-1K

本部分提供了ImageNet-1K实验的额外细节和结果。

C.1. 架构细节

我们为此任务使用了经典ResNet架构（He等，2016）的受限版本，该架构源自 `https://github.com/huyvnphan/PyTorch_CIFAR10`。与用于ImageNet的标准实现不同，这里的第一个卷积被约束为使用3×3的内核大小，而不是7×7。我们使用了ResNet-152结构，并在最终平均池化和投影到类别逻辑之前取输出。我们使用了大小为224×224的输入图像，这产生了用于交叉注意力中的键和值的14×14特征。

我们使用了以下超参数：

- $D = 4096$ (z^t和a^t的宽度)
- $k = 16$ (突触深度, 8 层向下和 8 层向上)
- $d_{\text{输入}} = 1024$ （注意力输出的宽度，o^t）
- n_{heads} {y*} 16
- 随机神经元选择配对（参见附录B.2）
- $D_{\text{out}} = 8196$ (宽度为 S^t_{out} 同步表示)
- $D_{\text{action}} = 2048$ (S^t_{action} 同步表示的宽度)
- $n_{\text{self}} = 32$ （用于恢复快照表示）
- $T = 50$ (内部点击)
- $M = 25$ (FIFO滚动记忆输入到NLMs)
- $d_{\text{隐藏}} = 64$（NLMs 内部 MLPs 的宽度）
- $p_{\text{dropout}} = 0.2$ (synapse模型的dropout概率)
- 没有位置嵌入

我们用于优化的设置如下：

- 使用 8 块 Nvidia H100 GPU，每块 GPU 的批量大小为 64 进行训练
- 500000次迭代用于训练，使用一种自定义采样方法，使得每个小批量可以有放回地采样

- AdamW (Loshchilov和Hutter, 2017)
- 学习率设为5e-4，并在10000个迭代进行线性预热，之后使用余弦退火学习率调度器衰减至零

- 梯度范数剪裁设置为20
- 无权重衰减

C.2. 损失函数

L列表 5 显示了用于训练 CTM 的图像分类损失函数的 Python 代码
我mageNet-1K. 这伴随在第2.5节中定义的损失。

C.3. 额外的演示 D. 2D 迷宫

D.1. 数据集

We 使用了迷宫数据集仓库生成了本次工作的迷宫：`https://github.com/understanding-search/maze-dataset`我们生成了大小为 19×19、39×39 和 99×99 的迷宫。

```python
def image_classification_loss(predictions, certainties, targets, use_most_certain=True):
    """
    Computes the maze loss with auto-extending cirriculum.

    Predictions are of shape: (B, class, internal_ticks),
    Certainties are of shape: (B, 2, internal_ticks),
        where the inside dimension (2) is [normalised_entropy, 1-normalised_entropy]
    Targets are of shape: [B]

    use_most_certain will select either the most certain point or the final point.
    """
    targets_expanded = torch.repeat_interleave(targets.unsqueeze(-1), predictions.size(-1), -1)
    # Losses are of shape [B, internal_ticks]
    losses = nn.CrossEntropyLoss(reduction='none')(predictions, targets_expanded)

    loss_index_1 = losses.argmin(dim=1)
    loss_index_2 = certainties[:,1].argmax(-1)
    if not use_most_certain:  # Revert to final loss if set
        loss_index_2[:] = -1

    batch_indexer = torch.arange(predictions.size(0), device=predictions.device)
    loss_minimum_ce = losses[batch_indexer, loss_index_1].mean()
    loss_selected = losses[batch_indexer, loss_index_2].mean()

    loss = (loss_minimum_ce + loss_selected)/2
    return loss
```

Listing 5 | Loss function implementation of Section 2.5 for standard classification tasks, used for ImageNet-1K.

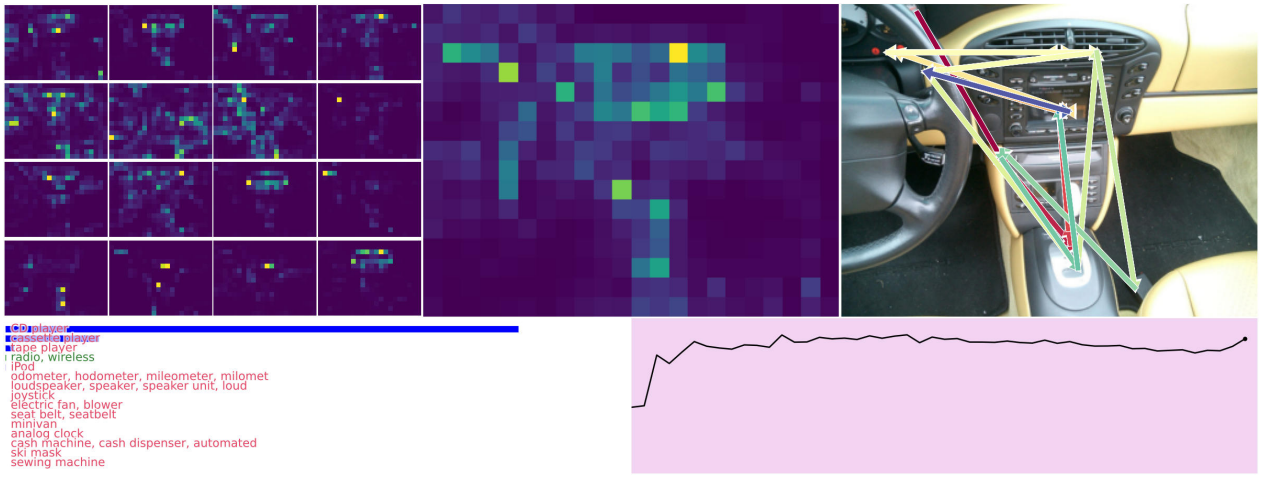


Figure 30 | Validation image index 1235, showing incorrect and uncertain prediction.

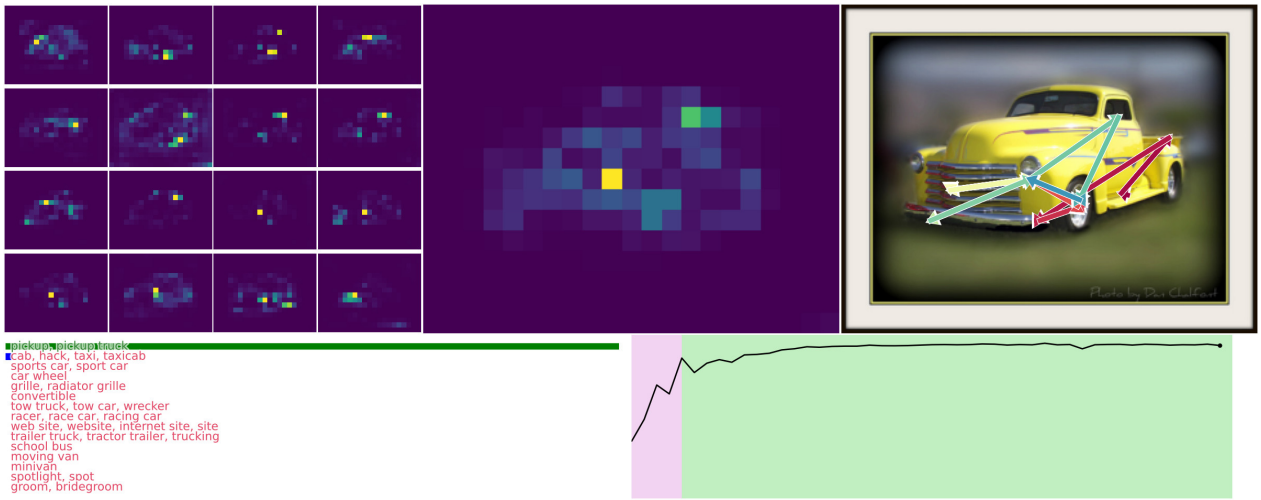


Figure 31 | Validation image index 15971, showing correct prediction and plausible 2nd most probable class.

```
def image_classification_loss(predictions, certainties, targets, use_most_certain=True):
    """
    Computes the maze loss with auto-extending cirriculum.

    Predictions are of shape: (B, class, internal_ticks),
    Certainties are of shape: (B, 2, internal_ticks),
        where the inside dimension (2) is [normalised_entropy, 1-normalised_entropy]
    Targets are of shape: [B]

    use_most_certain will select either the most certain point or the final point.
    """
    targets_expanded = torch.repeat_interleave(targets.unsqueeze(-1), predictions.size(-1), -1)
    # Losses are of shape [B, internal_ticks]
    losses = nn.CrossEntropyLoss(reduction='none')(predictions, targets_expanded)

    loss_index_1 = losses.argmin(dim=1)
    loss_index_2 = certainties[:,1].argmax(-1)
    if not use_most_certain:  # Revert to final loss if set
        loss_index_2[:] = -1

    batch_indexer = torch.arange(predictions.size(0), device=predictions.device)
    loss_minimum_ce = losses[batch_indexer, loss_index_1].mean()
    loss_selected = losses[batch_indexer, loss_index_2].mean()

    loss = (loss_minimum_ce + loss_selected)/2
    return loss
```

Listing 5 | 损失函数　　实施第2.5节中的标准分类任务　　，用于 ImageNet-1K。

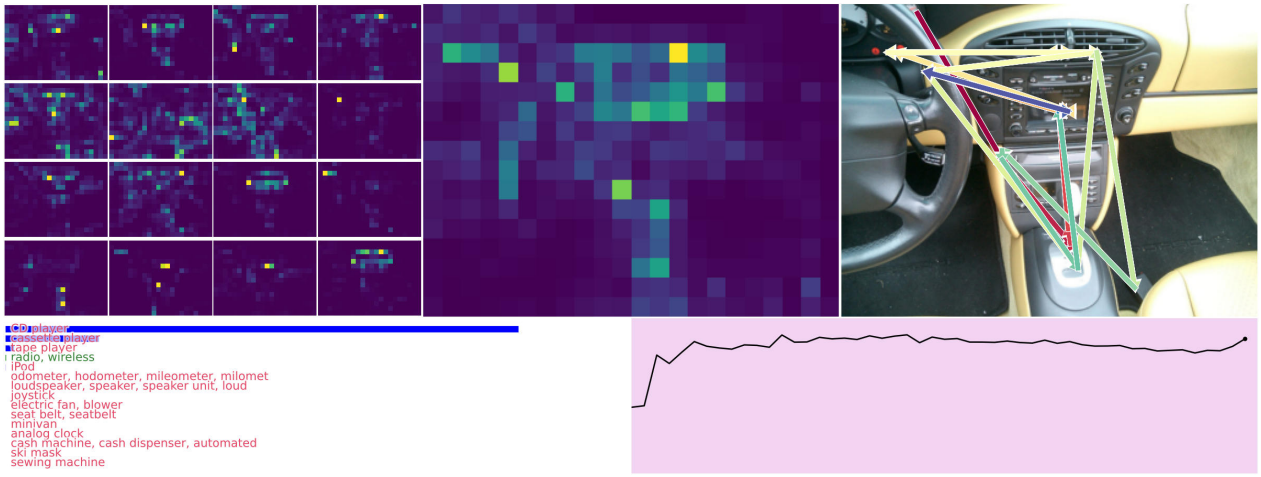


图30 | 验证图像索引1235，显示了错误和不确定的预测。

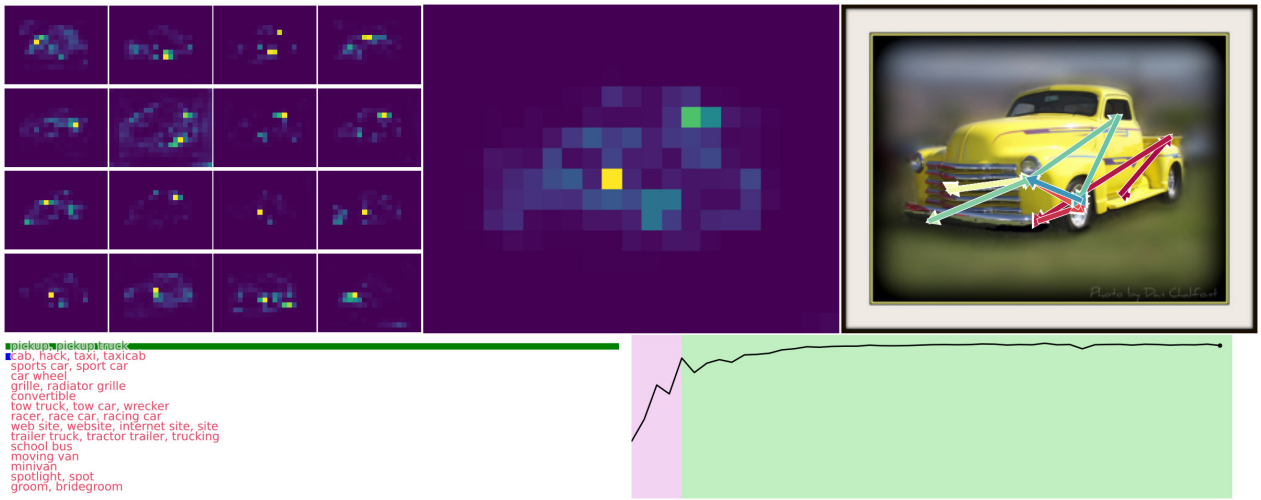


图31 | 验证图像索引15971，显示正确的预测和最可能的第二类。

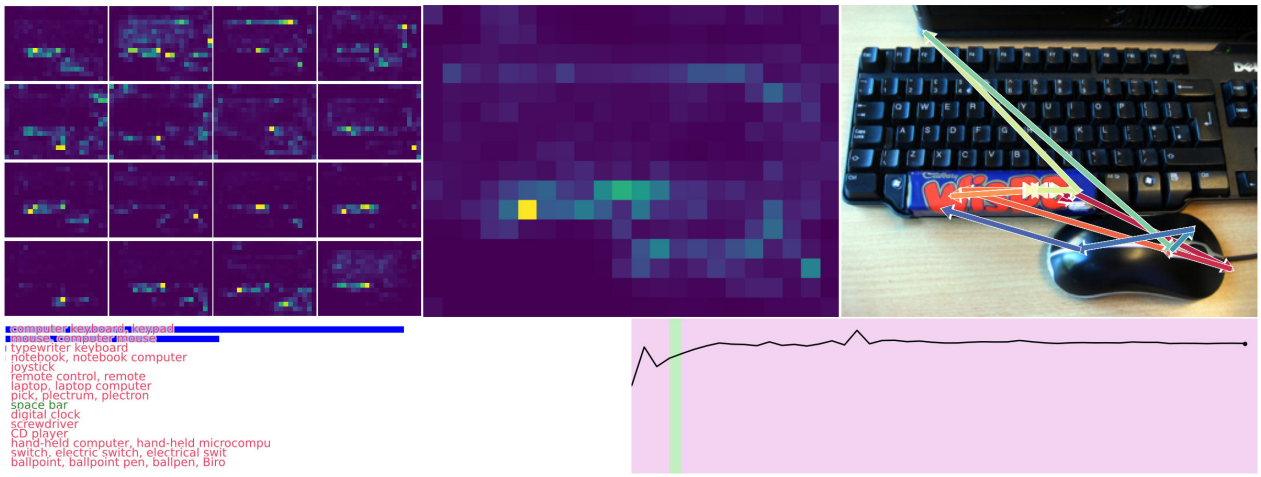


Figure 32 | Validation image index 21202, incorrect prediction after passing by correct prediction, showing ‘over-thinking’.

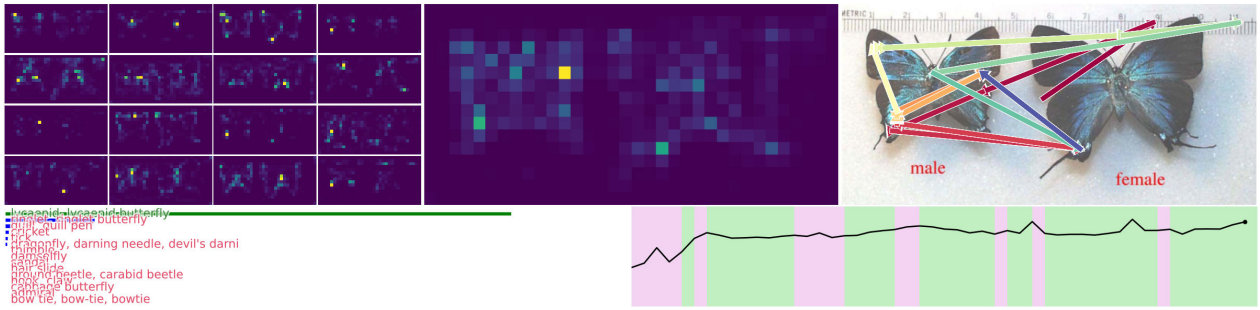


Figure 33 | Validation image index 39275, correct but uncertain prediction.

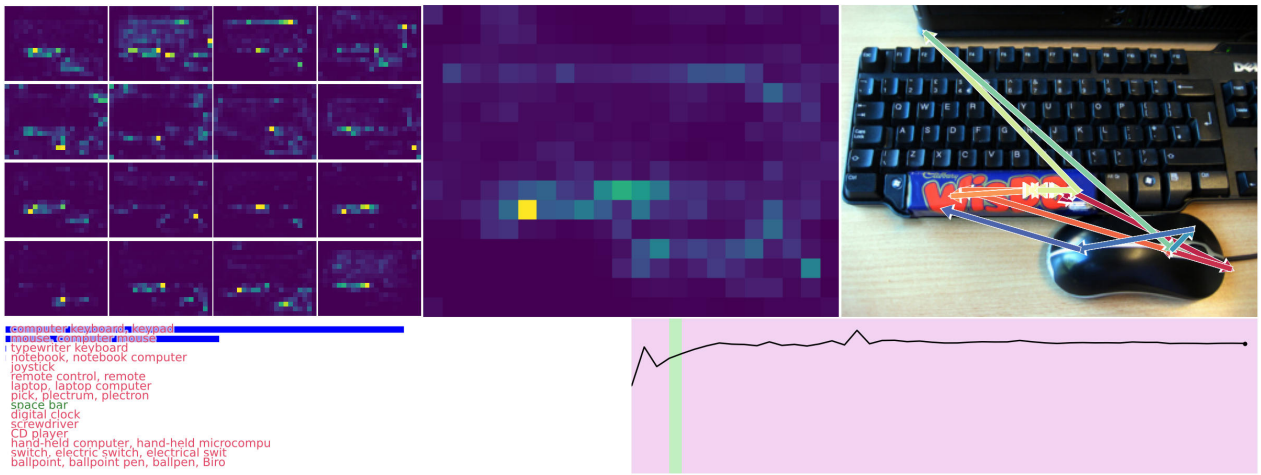


图32 | 验证图像索引21202，在正确预测后出现错误预测，显示“过度思考”。

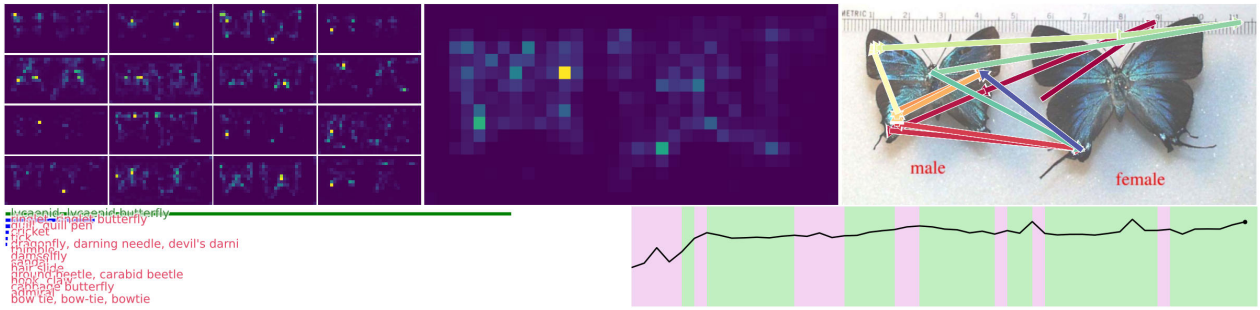


图33 | 验证图像索引39275，预测正确但不确定。

In each case we generated 50000 mazes and split them into train sets of size 45000 and test sets of size 5000. We used the 39×39 for training in this technical report and tested generalization on the 99×99. We provide all three maze datasets[8] in the CTM code repository.

D.2. Architecture details

We used the following hyperparameters:

- 39×39 mazes and a ResNet-34 backbone, where the keys and values were taken as features after the second hyper-block, resulting in a down-sample to 10×10
- $D = 2048$ (the width of $\mathbf{z}^t$ and $\mathbf{a}^t$)
- $k = 16$ (synapse depth, 8 layers down and 8 layers up)
- $d_{\text{input}} = 512$ (the width of attention output, $\mathbf{o}^t$)
- $n_{\text{heads}} = 16$
- Dense pairing for neuron selection (see Appendix B.2)
- $J_{\text{out}} = 32$ (width of $\mathbf{S}^t_{\text{out}}$ synchronization representation)
- $J_{\text{action}} = 32$ (width of $\mathbf{S}^t_{\text{action}}$ synchronization representation)
- $T = 75$ (internal ticks)
- $M = 25$ (FIFO rolling memory input to NLMs)
- $d_{\text{hidden}} = 32$ (width of MLPs inside NLMs)
- $p_{\text{dropout}} = 0.1$ (dropout probability for synapse model)
- No positional embedding

We used the following settings for optimization:

- Trained using a batch size of 64 on 1 H100 Nvidia GPU
- 1000000 iterations for training using AdamW (Loshchilov and Hutter, 2017)
- A learning rate of 1e-4 with a linear warmup of 10000 iterations and decaying to zero using a cosine annealing learning rate scheduler
- No weight decay

This resulted in a model with 31,998,330 parameters.

D.3. Maze curriculum

We adapted the loss function for solving mazes to include a curriculum element. Before computing t_1 and t_2 for the loss in Equation (12) we first altered the **loss at each internal tick** to only account for those steps in the maze that were correctly predicted with plus 5 additional steps along the path. This effectively lets the model (CTM or LSTM baselines) slowly learn to solve the maze from start to finish. Listing 6 shows how this is implemented when computing the loss and is used for both CTM and LSTM training.

D.4. Baselines details

We tested a number of LSTM baselines for solving this task, but struggled with stability during training (see Figure 7), particularly beyond a single LSTM layer or with a higher number of internal ticks. Hence we tested three LSTM configurations of depths 1,2, and 3. For each model we tested with 75 internal ticks to match the CTM, and 50 internal ticks for stability. We also tested a feed-forward model, projecting the feature space (before average pooling) into a hidden layer of the same width as the CTM, yielding a slightly higher parameter count. We kept all hyperparameters constant, yielding the following setups:

- **LSTM**, 1 layer, $T = 50$ and $T = 75$: 42,298,688

[8] We found the 19×19 beneficial for debugging, hence we provide it too.

我们在每种情况下，我们生成了50000个迷宫，并将它们分为大小为45000的训练集和测试集。
size 5000. 我们在这份技术报告中使用了 39×39 进行训练，并在测试中检验了泛化能力
99×99. 我们在CTM代码库中提供了所有三个迷宫数据集[8]。

D.2. 架构细节
我们使用了以下超参数：

- 39×39 迷宫和一个 ResNet-34 主干网，其中键和值取自第二个超块之后的特征，导致下采样到 10×10
- $D = 2048$ （z^t 和 a^t 的宽度）
- $k = 16$ (突触深度, 8 层向下和 8 层向上)
- $d_{\text{输入}} = 512$ （注意力输出的宽度，o^t）
- $n_{\text{heads}} = 16$
- 神经元选择的密集配对（参见附录 B.2）
- $J_{\text{out}} = 32$ (S^t_{out} 同步表示的宽度)
- $J_{\text{动作}} = 32$ （S^t_{action} 同步表示的宽度）
- $T = 75$ (内部点击)
- $M = 25$ (FIFO滚动记忆输入到NLMs)
- $d_{\text{隐藏}} = 32$（NLMs 内部 MLPs 的宽度）
- $p_{\text{dropout}} = 0.1$ (synapse模型的dropout概率)
- 没有位置嵌入

我们用于优化的设置如下：

- 使用批大小为 64 的 1 块 Nvidia H100 GPU 进行训练
- 1000000次迭代用于训练（使用AdamW，Loshchilov和Hutter, 2017）
- 学习率为 1e-4，带有 10000 个迭代的线性预热，并使用余弦退火学习率调度器衰减至零
- 无权重衰减

这 resulted 在一个具有 31,998,330 个参数的模型中。

D.3. 迷宫课程
我们将迷宫求解的损失函数进行了改编，加入了课程学习的元素。在计算方程(12)中的 t_1 和 t_2 之前，我们首先将每个内部时间步的损失修改为仅考虑那些正确预测的步骤，同时还包括路径上额外的5个步骤。这实际上让模型（CTM 或 LSTM 基线）能够逐步学会从起点到终点解决迷宫。第6列表示了在计算损失时如何实现这一点，并且这种方法同时用于CTM和LSTM的训练。

D.4. 基线细节
我们测试了多种LSTM基线来解决这个问题，但在训练过程中遇到了稳定性问题（参见图7），特别是在超过一层LSTM层或内部时钟数较多的情况下。因此，我们测试了深度分别为1、2和3的三种LSTM配置。对于每个模型，我们使用75个内部时钟与CTM匹配，并使用50个内部时钟以确保稳定性。我们还测试了一个前馈模型，将特征空间（在平均池化之前）投影到一个与CTM隐藏层宽度相同的隐藏层中，从而略微增加了参数数量。我们保持所有超参数不变，得到以下配置：

- LSTM, 1层, $T = 50$和$T = 75$: 42,298,688

[8]We found the 19×19 beneficial for debugging, hence we provide it too.

```python
def image_classification_loss(predictions, certainties, targets, use_most_certain=True):
    """
    Computes the maze loss with auto-extending cirriculum.

    Predictions are of shape: (B, class, internal_ticks),
    Certainties are of shape: (B, 2, internal_ticks),
        where the inside dimension (2) is [normalised_entropy, 1-normalised_entropy]
    Targets are of shape: [B]

    use_most_certain will select either the most certain point or the final point.
    """
    targets_expanded = torch.repeat_interleave(targets.unsqueeze(-1), predictions.size(-1), -1)
    # Losses are of shape [B, internal_ticks]
    losses = nn.CrossEntropyLoss(reduction='none')(predictions, targets_expanded)

    loss_index_1 = losses.argmin(dim=1)
    loss_index_2 = certainties[:,1].argmax(-1)
    if not use_most_certain:  # Revert to final loss if set
        loss_index_2[:] = -1

    batch_indexer = torch.arange(predictions.size(0), device=predictions.device)
    loss_minimum_ce = losses[batch_indexer, loss_index_1].mean()
    loss_selected = losses[batch_indexer, loss_index_2].mean()

    loss = (loss_minimum_ce + loss_selected)/2
    return loss
```

Listing 6 | Loss function implementation of Section 2.5 for maze route prediction, including an auto curriculum approach for both CTM and LSTM.

- **LSTM**, 2 layers, $T = 50$ and $T = 75$: 75,869,504 parameters
- **LSTM**, 3 layers, $T = 50$ and $T = 75$: 109,440,320 parameters
- **Feed-forward**, with a hidden layer width of 2048 (and GLU activation thereon): 54,797,632 parameters

D.5. Maze loss curves

Figure 34 gives the loss curves for the maze solving models in Section 4.1, showing how the CTM is more stable and performant when trained on this task.

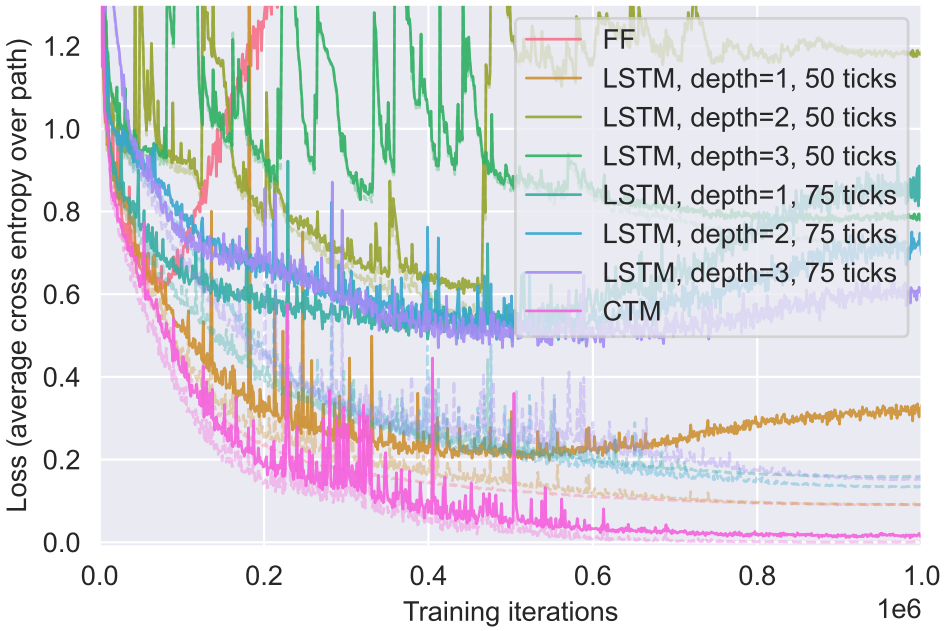


Figure 34 | Loss curves when training the CTM and baselines.

```
def image_classification_loss(predictions, certainties, targets, use_most_certain=True):
    """
    Computes the maze loss with auto-extending cirriculum.

    Predictions are of shape: (B, class, internal_ticks),
    Certainties are of shape: (B, 2, internal_ticks),
        where the inside dimension (2) is [normalised_entropy, 1-normalised_entropy]
    Targets are of shape: [B]

    use_most_certain will select either the most certain point or the final point.
    """
    targets_expanded = torch.repeat_interleave(targets.unsqueeze(-1), predictions.size(-1), -1)
    # Losses are of shape [B, internal_ticks]
    losses = nn.CrossEntropyLoss(reduction='none')(predictions, targets_expanded)

    loss_index_1 = losses.argmin(dim=1)
    loss_index_2 = certainties[:,1].argmax(-1)
    if not use_most_certain:  # Revert to final loss if set
        loss_index_2[:] = -1

    batch_indexer = torch.arange(predictions.size(0), device=predictions.device)
    loss_minimum_ce = losses[batch_indexer, loss_index_1].mean()
    loss_selected = losses[batch_indexer, loss_index_2].mean()

    loss = (loss_minimum_ce + loss_selected)/2
    return loss
```

L列表 6 | 第2.5节中用于迷宫路径预测的损失函数实现，包括自适应课程方法
f或两者CTM和LSTM。

- LSTM，2层，$T = 50$和$T = 75$: 75,869,504参数
- LSTM, 3层, $T = 50$和$T = 75$: 109,440,320参数
- 前馈网络，隐藏层宽度为2048（并在此处使用GLU激活）： 54,797,632个参数

D.5. 迷宫损失曲线

F图34给出了第4.1节中迷宫求解模型的损失曲线，展示了CTM是如何
m在针对这项任务训练时，它更加稳定和性能出色。

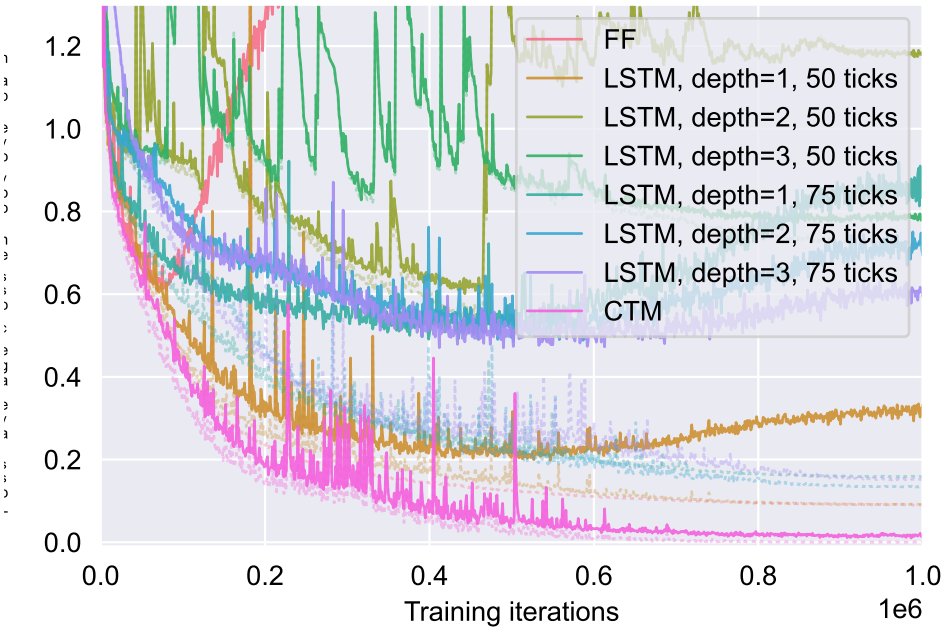


图34 | 训练CTM和基线时的损失曲线。

E. CIFAR-10 versus humans

We used a the first hyper-block of a constrained ResNet-18 backbone (see Appendix C.1): 5 convolutional layers in total and a downsample factor of 2×. We used the following hyperparameters for the CTM:

- $D = 256$ (the width of $\mathbf{z}^t$ and $\mathbf{a}^t$)
- $k = 10$ (synapse depth, 5 layers down and 5 layers up)
- $d_{\text{input}} = 64$ (the width of attention output, $\mathbf{o}^t$)
- $n_{\text{heads}} = 16$
- Random pairing for neuron selection (see Appendix B.2)
- $D_{\text{out}} = 256$ (width of $\mathbf{S}^t_{\text{out}}$ synchronization representation)
- $D_{\text{action}} = 512$ (width of $\mathbf{S}^t_{\text{action}}$ synchronization representation)
- $n_{\text{self}} = 0$
- $T = 50$
- $M = 15$ (FIFO rolling memory input to NLMs)
- $d_{\text{hidden}} = 64$ (width of MLPs inside NLMs)
- $p_{\text{dropout}} = 0.0$
- Weight decay of 0.0001
- No positional embedding

We used the following settings for optimization:

- Trained using a batch size of 512 on 1 H100 Nvidia GPU
- 600000 iterations for training using AdamW (Loshchilov and Hutter, 2017)
- A learning rate of 1e-4 with a linear warmup of 2000 iterations and decaying to zero using a cosine annealing learning rate scheduler

For the LSTM baseline a 2-layer LSTM was used as this performed better than a single layer LSTM setup and was relatively stable in training (compared to the maze task). The CTM synapse depth, memory length, and NLM hidden width were chosen such that the model width was kept constant (256) while parameter counts were closely matched between the CTM and LSTM. For the feed-forward model we kept the model width constant.

F. CIFAR-100

This section discusses the details of the CIFAR-100 experiments.

F.1. Architecture details

For Section 6.1 we used a the first two hyper-blocks of a constrained ResNet-34 backbone (see Appendix C.1): a downsample factor of 4×. For this experiment we varied D but kept all other hyperparameters as:

- $k = 8$ (synapse depth, 4 layers down and 4 layers up)
- $d_{\text{input}} = 512$ (the width of attention output, $\mathbf{o}^t$)
- $n_{\text{heads}} = 8$
- Random pairing for neuron selection (see Appendix B.2)
- $D_{\text{out}} = 2048$ (width of $\mathbf{S}^t_{\text{out}}$ synchronization representation)
- $D_{\text{action}} = 1024$ (width of $\mathbf{S}^t_{\text{action}}$ synchronization representation)
- $n_{\text{self}} = 32$
- $T = 50$
- $M = 25$ (FIFO rolling memory input to NLMs)
- $d_{\text{hidden}} = 32$ (width of MLPs inside NLMs)

E. CIFAR-10 versus 人类

我们使用了一个受限ResNet-18主干网络的第一个超块（参见附录C.1）：总共5个卷积层和下采样因子2×。我们为CTM使用的超参数如下：

- $D = 256$ (z^t和a^t的宽度)
- $k = 10$ (突触深度，5层向下和5层向上)
- $d_{\text{输入}} = 64$ （注意力输出的宽度，o^t）
- n_{heads} {y*} 16
- 随机神经元选择配对（参见附录B.2）
- $D_{\text{out}} = 256$ (S^t_{out} 同步表示的宽度)
- $D_{\text{action}} = 512$ (S^t_{action} 同步表示的宽度)
- $n_{\text{self}} = 0$
- $T = 50$
- $M = 15$ （FIFO滚动内存输入到NLMs）
- $d_{\text{隐藏}} = 64$（NLMs 内部 MLPs 的宽度）
- $p_{\text{dropout}} = 0.0$
- 权重衰减为0.0001
- 没有位置嵌入

我们用于优化的设置如下：

- 使用 1 块 Nvidia H100 GPU，批量大小为 512 进行训练
- 使用 AdamW（洛什科夫和赫特，2017）进行训练，共600000个迭代
- 学习率设为1e-4，并在2000个迭代进行线性预热，之后使用余弦退火学习率调度器衰减至零

对于LSTM基线，使用了2层LSTM，因为与单层LSTM相比，其表现更好且在训练过程中相对稳定（与迷宫任务相比）。CTM突触深度、记忆长度和NLM隐藏层宽度被选择为使模型宽度保持不变（256），同时CTM和LSTM的参数数量接近。对于前馈模型，保持模型宽度不变。

F. CIFAR-100

This section讨论了CIFAR-100实验的细节。

F.1. 架构细节

F或在第6.1节中，我们使用了受限ResNet-34主干网络的前两个超块（参见
A附录 C.1): 下采样因子为 4×。对于这个实验，我们变化了D，但保持了所有其他
h超参数为：

- $k = 8$ (突触深度, 4层向下和4层向上)
- $d_{\text{输入}} = 512$ （注意力输出的宽度，o^t）
- $n_{\text{heads}} = 8$
- 随机神经元选择配对（参见附录B.2）
- $D_{\text{out}} = 2048$ (S^t_{out} 同步表示的宽度)
- $D_{\text{action}} = 1024$ (S^t_{action} 同步表示的宽度)
- $n_{\text{self}} = 32$
- $T = 50$
- $M = 25$ (FIFO滚动记忆输入到NLMs)
- $d_{\text{隐藏}} = 32$ (NLMs内部MLPs的宽度)

- $p_{\text{dropout}} = 0.2$
- No weight decay
- No positional embedding

For Section 6.2 we used the first two hyper-blocks of a constrained ResNet-19 backbone (see Appendix C.1): a downsample factor of 4×. For this experiment we varied T but kept all other hyperparameters as:

- $D = 512$
- $k = 4$ (synapse depth, 2 layers down and 2 layers up)
- $d_{\text{input}} = 256$ (the width of attention output, $\mathbf{o}^t$)
- $n_{\text{heads}} = 4$
- Random pairing for neuron selection (see Appendix B.2)
- $D_{\text{out}} = 256$ (width of $\mathbf{S}^t_{\text{out}}$ synchronization representation)
- $D_{\text{action}} = 256$ (width of $\mathbf{S}^t_{\text{action}}$ synchronization representation)
- $n_{\text{self}} = 0$
- $M = 25$ (FIFO rolling memory input to NLMs)
- $d_{\text{hidden}} = 16$ (width of MLPs inside NLMs)
- $p_{\text{dropout}} = 0.0$
- Weight decay of 0.001
- No positional embedding

This model was set up to be more constrained compared to the other CIFAR-100 ablation because of the overhead induced by using more ticks. We explained in Section 6.2 that the variants trained with longer ticks could benefit from more training, and this disparity would be greater should we use bigger models for this experiment.

G. Parity

G.1. Dataset details

The input data for the parity task is a vector of length 64, where at each position is either a −1 or 1. The target for each sample is a vector of the same size, where at each position is the parity of the sequence up to that position, which we refer to as the cumulative parity. This data is generated on the fly, each time a new batch is fetched.

G.2. Architecture details

The following architecture is used for the experiments in the parity task. Inputs to the model are of shape $(B, 64)$, where the size of the minibatch and sequence length are B and 64, respectively. Each of the values in the 64-length sequence are either -1 or 1, at random. First, the values of -1 and 1 are converted into embeddings in $d_{\text{embed}} = d_{\text{input}}$ and positional embeddings are added. The resulting embeddings are passed through a linear layer with layer normalization to form (identical) attention keys and values. As described in section 2, the synchronization between J_{action} neurons is computed and from this representation an attention query is formed. This query is then used to compute the attention values, which are concatenated to the activated state to be processed by the synapses and the neuron-level models. For the synapses we use a shallow feedforward network. This process repeats for T internal ticks, where at each internal tick t, the synchronization can be computed between J_{out} neurons and projected to the logit space.

In the parity task, we experimented with how the model performs with a varying number of internal ticks and memory length. As a baseline, we use single-layer LSTMs which are both parameter matched and use the same number of internal ticks as the CTM.

- $p_{\text{dropout}} = 0.2$
- 无权重衰减
- 没有位置嵌入

对于第6.2节，我们使用了受限ResNet-19主干网络的前两个超块（参见附录C.1）：下采样因子为4×。对于这个实验，我们变化了T，而保持其他所有超参数为：

- $D = 512$
- $k = 4$ (突触深度, 2层向下和2层向上)
- $d_{\text{输入}} = 256$ （注意力输出的宽度，o^t）
- $n_{\text{heads}} \{y^*\} 4$
- 随机神经元选择配对（参见附录B.2）
- $D_{\text{out}} = 256$ (S^t_{out} 同步表示的宽度)
- $D_{\text{action}} = 256$ (S^t_{action} 同步表示的宽度)
- $n_{\text{self}} = 0$
- $M = 25$ (FIFO滚动记忆输入到NLMs)
- $d_{\text{隐藏}} = 16$ (NLMs内部MLPs的宽度)
- $p_{\text{dropout}} = 0.0$
- 权重衰减为0.001
- 没有位置嵌入

这个模型被设置得比其他CIFAR-100消融实验更加受限，因为使用更多刻度产生了额外开销。我们在第6.2节中解释过，使用更长刻度训练的变体可以从更多的训练中受益，如果我们为此实验使用更大的模型，这种差异会更大。

G. 奇偶性

G.1. 数据集详情

The input data for the parity任务是长度为64的向量，在每个位置处是1或−1。每个样本的目标是大小相同的向量，在每个位置处是该位置之前序列的奇偶校验和，我们称之为累积奇偶校验。这些数据是实时生成的，每次获取新批次时都会生成。

G.2. 架构细节

以下架构用于奇偶任务的实验。模型的输入形状为$(B, 64)$，其中小批量大小和序列长度分别为B和64。序列中的每个值随机为-1或1。首先，将-1和1的值转换为嵌入，并添加位置嵌入。结果嵌入通过具有层归一化的线性层形成相同的注意力键和值。如第2节所述，计算J_{action}神经元之间的同步，并从该表示中形成注意力查询。然后使用该查询计算注意力值，将这些值连接到激活状态，由突触和神经级模型处理。对于突触，我们使用浅层前馈网络。此过程在T个内部时钟周期内重复，其中在每个内部时钟周期t，可以计算J_{out}个神经元之间的同步并投影到逻辑空间。

我在奇偶性任务中，我们研究了模型在不同数量的内部
ticks 和记忆长度。作为基线，我们使用参数匹配的单层LSTMs
a使用与CTM相同的内部时钟数。

All CTM models share a common set of architectural hyperparameters, listed below. The Table 3 shows the subset of hyperparameters that vary across experimental configurations.

- $d_{\text{model}} = 1024$
- $d_{\text{input}} = 512$
- $d_{\text{hidden}} = 4$
- $k = 1$
- $p_{\text{dropout}} = 0$
- $n_{\text{heads}} = 8$
- $J_{\text{action}} = 32$
- $J_{\text{out}} = 32$
- Semi-dense pairing was used for selecting neurons for synchronization
- Absolute positional encoding was added to the input features

Model	T	M	d_{model}	**Total Parameters**
CTM	1	1	1024	4908706
LSTM	1	–	669	4912710
CTM	10	5	1024	5043874
LSTM	10	–	686	5050716
CTM	25	10	1024	5212834
LSTM	25	–	706	5224386
CTM	50	25	1024	5719714
LSTM	50	–	765	5722374
CTM	75	25	1024	5719714
LSTM	75	–	765	5722374
CTM	100	50	1024	6564514
LSTM	100	–	857	6567486

Table 3 | Model hyperparameters for the parity task (that vary across configurations).

G.3. Optimization details

The CTM was trained using the certainty-based loss function described in section 2.5, whereas the LSTM baselines utilized the cross-entropy loss computed at the final internal tick. This choice was made due to initial difficulties in training the LSTM effectively with the certainty-based loss. In Figure 35, we compare training accuracy curves for the LSTM baselines with 10 and 25 iterations, trained with either the final or certainty-based loss. Generally, both loss functions lead to unstable training for LSTMs with multiple internal ticks.

We used the following settings for optimization:

- Trained using a batch size of 64 on 1 H100 Nvidia GPU.
- 200000 iterations for training using AdamW (Loshchilov and Hutter, 2017).
- A learning rate of 1e-4 with a linear warmup of 500 iterations and decaying to zero using a cosine annealing learning rate scheduler.

G.4. Results

Model performance varies significantly between seeds Figure 19 shows the accuracy over training for various CTM and LSTM configurations, with each configuration averaged over three independent runs. These training curves exhibit considerable variance due to significant differences in performance between runs, strongly influenced by the initial random seed. For example, Figure 36 shows individual

A所有 CTM 模型共享一组共同的架构超参数，如下表所示。表 3
s如何关于实验配置中变化的超参数子集。

- $d_{\text{模型}} = 1024$
- $d_{\text{输入}} = 512$
- $d_{\text{隐藏}} = 4$
- $k = 1$
- $p_{\text{dropout}} = 0$
- n_{heads} {v*} 8
- $J_{\text{动作}} = 32$
- $J_{\text{out}} = 32$
- 半密集配对用于选择同步神经元
- 绝对位置编码被添加到输入特征中

Model	T	M	d_{model}	**Total Parameters**
CTM	1	1	1024	4908706
LSTM	1	–	669	4912710
CTM	10	5	1024	5043874
LSTM	10	–	686	5050716
CTM	25	10	1024	5212834
LSTM	25	–	706	5224386
CTM	50	25	1024	5719714
LSTM	50	–	765	5722374
CTM	75	25	1024	5719714
LSTM	75	–	765	5722374
CTM	100	50	1024	6564514
LSTM	100	–	857	6567486

表 3 | 模型超参数（在不同配置中变化）用于奇偶任务

G.3. 最优化细节

CTM 使用了第 2.5 节中描述的基于 certainty 的损失函数进行训练，而 LSTM 基线则使用在最终内部时钟处计算的交叉熵损失。这一选择是因为最初在使用基于 certainty 的损失训练 LSTM 时遇到了有效训练的困难。在图 35 中，我们比较了使用 10 次和 25 次迭代训练的 LSTM 基线的训练准确率曲线，这些基线分别使用最终损失或基于 certainty 的损失进行训练。总体而言，这两种损失函数都会导致具有多个内部时钟的 LSTM 的训练不稳定。

我们用于优化的设置如下：

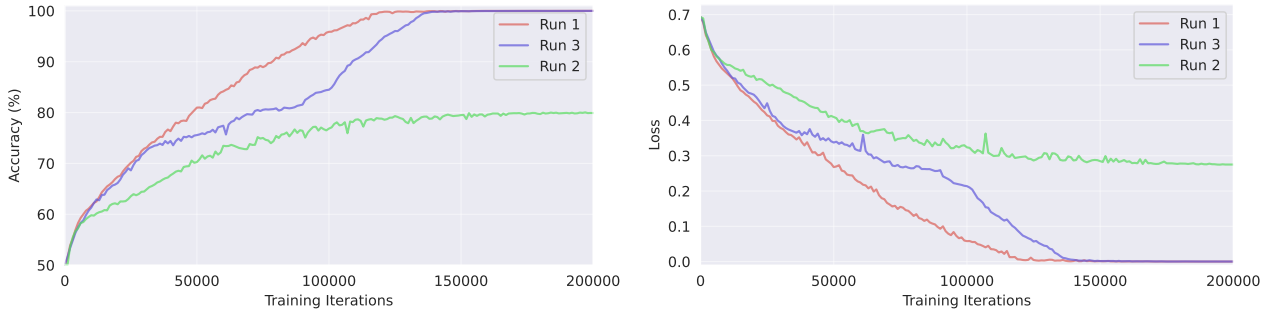
- 使用 1 块 Nvidia H100 GPU，批量大小为 64 进行训练。
- 200000次迭代用于训练（使用AdamW，Loshchilov和Hutter, 2017）。
- 一个学习率为 1e-4，使用 500 个迭代的线性预热，并使用余弦退火学习率调度器衰减至零。

G.4. 结果

模型性能在不同的种子之间差异显著。图19显示了各种CTM和LSTM配置在训练过程中准确率的变化，每种配置在三次独立运行上的平均值。这些训练曲线由于每次运行之间性能的巨大差异而表现出显著的波动，这些差异受到初始随机种子的显著影响。例如，图36显示了个体

Figure 35 | Test accuracies for LSTM baselines, trained with either the certainty-based loss or the cross-entropy loss at the final internal tick. Both loss functions lead to unstable learning.

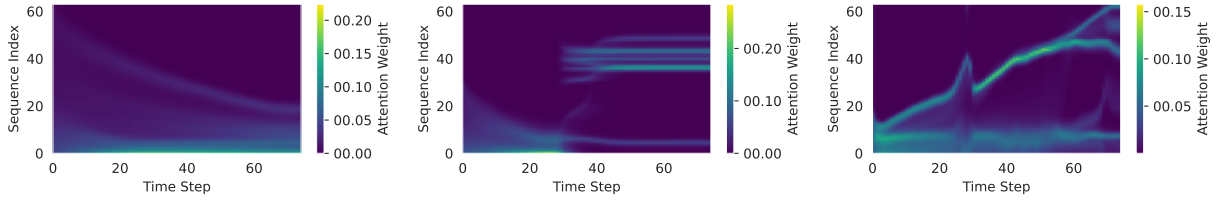
training curves for the CTM trained with 75 internal ticks and a memory length of 25. Runs 1 and 3 reach perfect accuracy, while run 2 converges to a suboptimal solution.



(a) Accuracy. **(b)** Loss.

Figure 36 | Training curves for three CTMs trained with three random seeds. Run 1 and 3 converge to a loss of zero, while the other run converges to a non-zero loss.

Furthermore, all three of these models display significantly different behaviors, with each CTM attending to very different parts of the input sequence over the 75 internal ticks. These attention patterns over the internal ticks are shown in Figure 37. Run 3 results in a model that attends from the beginning to the end of the entire sequence, while run 1 attends in reverse order.



(a) Run 1. **(b)** Run 2. **(c)** Run 3.

Figure 37 | Attention patterns for each of the three runs after training.

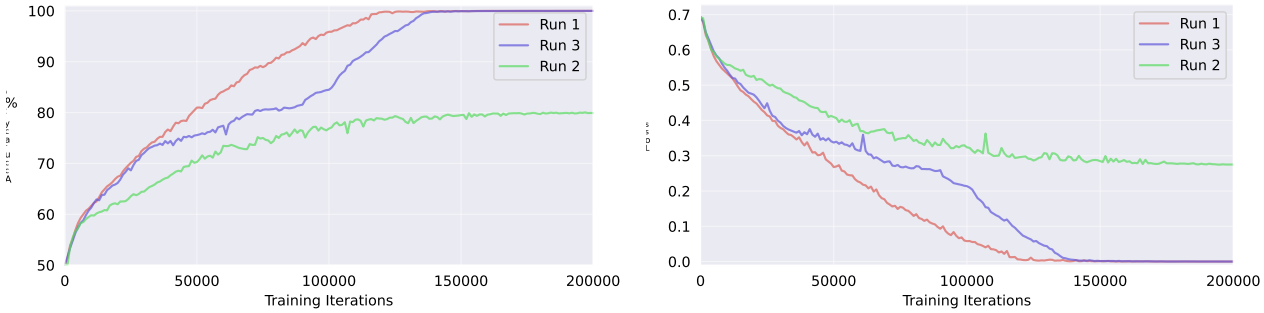
H. Q&A MNIST

H.1. Architecture details

Unlike other tasks, the Q&A MNIST task processes multiple input types: MNIST digit images, embeddings for operator and index markers, and zero tensors as answer flags. MNIST images undergo

图35 | 在最终内部时间步长处使用基于 certainty 的损失或交叉熵损失训练的 LSTM 基线的测试准确率。两种损失函数都导致学习不稳定。

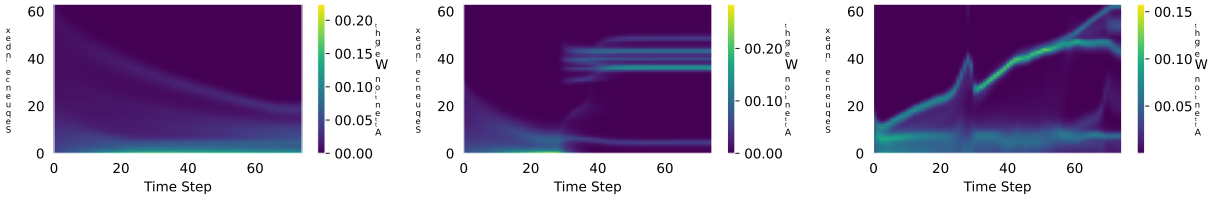
t为使用75个内部节拍和记忆长度为25训练的CTM训练的降雨曲线。运行1和3
r每个完美准确度，而在运行2中收敛到一个次优解。



(a) Accuracy. **(b)** Loss.

图36 | 三种CTM使用三个随机种子训练的训练曲线。运行1和3收敛到损失为零，而其他运行收敛到非零损失。

此外，这三种模型显示了显著不同的行为，在75个内部节拍中，每个CTM都非常关注输入序列的不同部分。这些内部节拍上的注意力模式如图37所示。运行3的结果是从整个序列的开始到结束进行关注，而运行1则是逆序关注。



(a) Run 1. **(b)** Run 2. **(c)** Run 3.

图37 | 每次训练后的三种注意模式。

H. Q&A MNIST

H.1. 架构细节

U不同于其他任务，Q&A MNIST 任务处理多种输入类型：MNIST 数字图像，em-
b操作符和索引标记的填充，以及零张量作为答案标志。MNIST图像 undergo

preprocessing through a convolutional backbone consisting of two convolutional blocks, each containing a convolutional layer, batch normalization, a ReLU activation, and a max pooling layer. Outputs from this backbone form attention keys and values, which the CTM queries using projections from the synchronization representation. The resulting attention outputs are concatenated with the CTM's activated state before synaptic processing. In contrast, operator and index embeddings, as well as answer flags, bypass the convolutional backbone and attention mechanism, being directly concatenated to the CTM's activated state. Operators use learned embeddings, indices utilize sinusoidal embeddings (Vaswani et al., 2017), and answer flags are zero vectors matching the embedding dimension.

For comparison, parameter and internal tick matched single-layer LSTM baselines were used. The common parameters used in the experiment are as follows:

- $d_{\text{model}} = 1024$
- $d_{\text{input}} = 64$
- $d_{\text{hidden}} = 16$
- $k = 1$
- $p_{\text{dropout}} = 0$
- $n_{\text{heads}} = 4$
- $J_{\text{action}} = 32$
- $J_{\text{out}} = 32$
- Semi-dense pairing was used for selecting neurons for synchronization.
- Positional encoding was not used.

Detailed hyperparameters of the CTM are provided in Table 4.

Model	T	M	Repeats/Input	Answering Steps	Total Parameters
CTM	1	3	1	1	2,501,388
LSTM	1	–	1	1	2,507,218
CTM	10	30	10	10	3,413,772
LSTM	10	–	10	10	3,418,954

Table 4 | Differing model hyperparameters and total parameters for the Q&A MNIST experiments. The column **Repeats/Input** refers to the number of internal ticks the model used to process a unique input. For example, **Repeats/Input** = 10 implies that 10 internal ticks are used to process each MNIST digit and each index or operator embedding. The **Answering Steps** refers to the number of internal ticks the answering flag is observed for.

H.2. Optimization details

The CTM was trained using the certainty-based loss function described in section 2.5, while the LSTM baselines were trained using the cross-entropy loss at the final internal tick. We used the following settings for optimization:

- Trained using a batch size 64 on 1 H100 Nvidia GPU.
- 300000 iterations for training using AdamW (Loshchilov and Hutter, 2017).
- A learning rate of 1e-4 with a linear warmup of 500 iterations and decaying to zero using a cosine annealing learning rate scheduler.

I. Reinforcement learning

I.1. Environment details

CartPole. The CartPole task (*CartPole-v1*) is a classic task in reinforcement learning. It involves balancing a pole, which is hinged to a cart moving along a frictionless track. The system is controlled

预处理通过一个卷积骨干，该骨干由两个卷积块组成，每个块包含一个卷积层、批量归一化、ReLU激活和最大池化层。该骨干的输出形成注意力键和值，CTM 使用同步表示的投影对其进行查询。结果的注意力输出与CTM 的激活状态在突触处理前进行连接。相比之下，操作符和索引嵌入以及答案标志绕过卷积骨干和注意力机制，直接与CTM 的激活状态进行连接。操作符使用学习到的嵌入，索引使用正弦嵌入（Vaswani等，2017），而答案标志是与嵌入维度匹配的零向量。

F或者比较，使用了参数和内部时钟匹配的单层LSTM基线。
c实验中常用的参数如下：

- $d_{\text{模型}} = 1024$
- $d_{\text{输入}} = 64$
- $d_{\text{隐藏}} = 16$
- $k = 1$
- $p_{\text{dropout}} = 0$
- n_{heads} {v*} 4
- $J_{\text{动作}} = 32$
- $J_{\text{out}} = 32$
- 半密集配对用于选择同步的神经元。
- 位置编码未被使用。

CTM的详细超参数见表4。

Model	T	M	Repeats/Input	Answering Steps	Total Parameters
CTM	1	3	1	1	2,501,388
LSTM	1	–	1	1	2,507,218
CTM	10	30	10	10	3,413,772
LSTM	10	–	10	10	3,418,954

表 4 | Q&A MNIST 实验中不同模型超参数和总参数。列 Repeat-s/Input 表示模型处理每个唯一输入时使用的内部时钟数。例如，Repeats/Input = 10 表示每处理一个 MNIST 数字以及每个索引或操作嵌入时使用 10 个内部时钟。Answering Steps 表示观察回答标志所需的内部时钟数。

H.2. 最优化细节

T他使用了在第2.5节中描述的基于 certainty 的损失函数来训练CTM，而LSTM
b边界线是在最终内部时钟处使用交叉熵损失进行训练的。我们使用了以下方法
s优化设置：

- 训练使用批大小为 64 的 1 块 Nvidia H100 GPU。
- 300000次迭代用于训练（使用AdamW，Loshchilov和Hutter, 2017）。
- 一个学习率为 1e-4，使用 500 个迭代的线性预热，并使用余弦退火学习率调度器衰减至零。

I. 强化学习

I.1. 环境详情

CartPole. The CartPole 任务(*CartPole-v1*)是强化学习中的一个经典任务。它涉及
b平衡一个铰接在小车上并沿无摩擦轨道移动的杆。该系统受到控制

by applying horizontal forces (left or right) to the cart, with the objective of keeping the pole upright. A reward of +1 is given for every step taken, with the episode terminating when either the pole angle is greater than $\pm 12^\circ$, the cart position is greater than ± 2.4, or the episode length is greater than the maximum number of steps, which we set to 200 steps. Furthermore, we use reward normalization, such that the exponential moving average of immediate rewards has an approximately fixed variance.

The action space is composed of 2 discrete actions, corresponding to the direction of force applied to the cart. In the typical cartpole task, the observation space is of shape $(4,)$ with values corresponding to the cart position, cart velocity, pole angle and pole angular velocity. For our experiments with the CTM, however, the cart velocity and the pole angular velocity are masked to make the environment partially observable.

Acrobot. In the Acrobot task (*Acrobot-v1*), a system composed of two links connected linearly to form a chain, with one end of the chain fixed. The joint between the two links is actuated, while the joint at the fixed end is free to rotate. The goal is to apply torques to the actuated joint to swing the free end of the chain above a certain height in as few steps as possible, with the chain initially hanging down with some initial random angle and velocity. An episode ends when the chain reaches above the required height, or a maximum number of steps are taken, which we set to 500 steps. A reward of -1 is incurred for all steps taken to reach the goal, with a limit of -100.

The action space consists of three discrete actions, which are to apply -1, 0 and 1 Nm of torque to the actuated joint. The observation space is of shape (6,) composed of the sine and cosine of the angle made by the first joint (such that an angle of 0 indicated the first link is pointing directly downwards), the sine and cosine of the second link (such that an angle of 0 corresponds to having the same angle between the two links), and the angular velocity of two angles. As in the CartPole task, these two angular velocity components are masked such that the environment is only partially observable.

MiniGrid Four Rooms. The MiniGrid Four Rooms task (*MiniGrid-FourRooms-v0*) is a reinforcement learning environment in which an agent must navigate in a grid world composed of four rooms interconnected by four gaps in the walls. The agent receives a reward of $1 - 0.9(\text{step count} \times \text{max steps})$ if it reaches the goal located at the green square, or 0 otherwise. Again, reward normalization is used to normalize the moving average of past rewards. The agent's position, the goal position, and each of the four gaps in the walls are positioned randomly at the start of each episode. The environment terminates when the agent reaches the goal or when the maximum number of steps has been reached, which we set to 300 steps.

The action space is composed of 7 discrete actions, corresponding to turn left, turn right, go forward, pickup, drop, toggle, and done. Of these actions, only the first three are relevant to the task, with the other four actions behaving as a wait or no-op action. In this task, the agent has a limited field of view, consisting of the 7×7 grid located in front of the agent. The observation space of this environment is of shape $7 \times 7 \times 3$, where each of the 7×7 tiles is encoded as a three-dimensional tuple corresponding to the object, color and state IDs at that position. Specifically, there are 11 different objects (such as wall, floor, etc.), 6 different colors and 3 different states (open, closed, etc.).

I.2. Architecture details

The configuration of the CTM for training with PPO is as follows. First, observations are processed by a backbone, such as a feedforward network, and are concatenated to the current activated state of the CTM, without an attention mechanism, for processing over a fixed number of internal ticks. After this fixed number of internal ticks, the synchronization between the output neurons is calculated and passed to the actor and critic heads for selecting the next action and estimating the state value.

通过向小车上施加水平力（左或右），以使杆保持直立。每走一步奖励 +1，当杆的角度大于 ±12° 、小车的位置大于 ±2.4，或者步数超过最大步数（我们设为200步）时，该episode结束。此外，我们使用奖励归一化，使得即时奖励的指数移动平均值具有大约固定的方差。

动作空间由2个离散动作组成，对应于施加在小车上的力的方向。在典型的倒立摆任务中，观察空间的形状为(4,)，值对应于小车位置、小车速度、杆角度和杆角速度。然而，在我们对CTM的实验中，小车速度和杆角速度被屏蔽，使得环境部分可观测。

Acrobot. 在Acrobot任务(*Acrobot-v1*)中，由两个线性连接的链接组成一个链，一端固定。两个链接之间的关节可驱动，而固定端的关节可以自由旋转。目标是通过对可驱动关节施加扭矩，尽可能在较少的步骤内将链的自由端抬高到某个高度以上，链最初以某些初始随机角度和速度下垂。当链达到所需高度或达到最大步数（我们设置为500步）时，一集结束。对于所有达到目标所需的步骤，会获得–1的奖励，奖励上限为–100。

动作空间由三个离散动作组成，分别是施加到驱动关节上的扭矩 −1, 0 和 1 Nm。观测空间的形状为 (6,)，由第一个关节所形成角度的正弦和余弦（角度为 0 表示第一个连杆直接向下），第二个连杆的正弦和余弦（角度为 0 表示两个连杆之间的角度相同），以及两个角度的角速度组成。与 CartPole 任务类似，这两个角速度分量被遮蔽，使得环境只能部分可观测。

MiniGrid 四个房间。MiniGrid 四个房间任务(*MiniGrid-FourRooms-v0*)是一个强化学习环境，在这个环境中，智能体必须在一个由四个相连的房间组成的网格世界中导航。智能体如果到达位于绿色方块的目标位置，将获得一个奖励，该奖励等于1−0.9(步数×最大步数)，否则获得0。同样，使用奖励归一化来归一化过去奖励的移动平均值。智能体的位置、目标位置以及四个墙壁缺口的位置在每个回合开始时随机放置。当智能体到达目标或达到最大步数（我们设为300步）时，环境终止。

动作空间由7个离散动作组成，对应左转、右转、前进、拾取、放下、切换和完成。在这7个动作中，只有前三个与任务相关，其余四个动作表现为等待或无操作动作。在这个任务中，代理的视野有限，由代理前方的7×7网格组成。此环境的观测空间形状为7×7×3，其中每个7×7的格子编码为一个三维元组，对应于该位置的物体、颜色和状态ID。具体来说，有11种不同的物体（如墙壁、地板等），6种不同的颜色和3种不同的状态（如打开、关闭等）。

I.2. 架构细节

CTM在使用PPO训练时的配置如下。首先，观察数据通过一个主干，例如前馈网络，并与CTM当前激活的状态连接起来，不使用注意力机制，在固定数量的内部时钟周期内进行处理。在经过这个固定数量的内部时钟周期后，输出神经元之间的同步计算并传递给行为头和批评头，以选择下一个动作并估计状态值。

Unlike the other tasks, which calculate synchronization across the entire activation history, in the RL setting we use a sliding window of size memory length M. This approach prevents the buildup of very long activation histories, which may grow into the thousands for these tasks. Additionally, this allows for maintaining tensors of the same shape across all stages of the episode rollouts. To facilitate this, the CTM is initialized with both a learned initial state trace and a learned initial activated state trace, which are supplied to the model on the initialization of each episode. After a single forward pass of the model (corresponding to a single environment step), these state traces will be maintained and provided to the model on the next environment step. This allows the CTM to process a continuous history of activity, enabling activations from many environment states in the past to impact the present.

For the classic control tasks, the observation backbone is composed of two blocks containing a linear layer, a gated linear unit (GLU) (Dauphin et al., 2017) and layer normalization. A similar input processing is carried out for the navigation task, however, each of the object, color and state IDs are first embed in $d_{embed} = 8$. While for the CTM the output of the backbone is concatenated to the current activated state, for the LSTM baseline, the output is instead processed for the same number of internal ticks as the CTM. The actor and critic heads are implemented as two-layer multilayer perceptrons (MLPs), each comprising two hidden layers of 64 neurons with ReLU activations. For the CTM, these heads receive the synchronization of the output neurons as inputs, while for the LSTM baselines, they receive the hidden state of the LSTM after T internal tick. Dense pairing was used to select neurons for synchronization.

Unlike other tasks such as image classification (Section 3), which use a UNet-style synapse model, the RL tasks employ a two-layer feedforward synapse, where each layer consists of a linear transformation, a GLU and LayerNorm. Empirically, we found that two of these layers significantly outperformed a single layer, particularly in the navigation task, where a single-layer synapse consistently failed to match the LSTM's average episode length.

The model hyperparameters used for the experiments for CartPole, Acrobot and MiniGrid Four Rooms can be found in Tables 5 to 7.

Model	T	M	d_{model}	d_{input}	d_{hidden}	J_{out}	**Total Parameters**
CTM	1	10	128	128	4	16	175437
LSTM	1	–	118	128	–	–	175855
CTM	2	20	128	128	4	16	188237
LSTM	2	–	126	128	–	–	188863
CTM	5	50	128	128	4	16	226637
LSTM	5	–	148	128	–	–	227275

Table 5 | Model hyperparameters for the CartPole experiments.

Model	T	M	d_{model}	d_{input}	d_{hidden}	J_{out}	**Total Parameters**
CTM	1	5	256	64	4	16	350094
LSTM	1	–	243	64	–	–	350118
CTM	2	10	256	64	4	16	362894
LSTM	2	–	249	64	–	–	364290
CTM	5	25	256	64	4	16	401294
LSTM	5	–	265	64	–	–	403490

Table 6 | Model hyperparameters for the Acrobot experiments.

与其他任务不同，这些任务计算整个激活历史的同步性，而在RL设置中，我们使用大小为记忆长度M的滑动窗口。这种方法可以防止激活历史变得非常长，这些任务的激活历史可能会增长到数千个。此外，这使得在所有阶段的episode展开中保持相同形状的张量成为可能。为了实现这一点，CTM在每个episode初始化时都使用一个学习到的初始状态轨迹和一个学习到的初始激活状态轨迹，这些轨迹在每个episode的初始化时提供给模型。在模型进行一次前向传递（对应于一个环境步骤）之后，这些状态轨迹将被维护并在下一个环境步骤中提供给模型。这使得CTM能够处理连续的活动历史，从而使过去的许多环境状态的激活影响当前的状态。

对于经典控制任务，观测主干由包含线性层、门控线性单元（GLU）(Dauphin等，2017) 和层规范化两个块组成。对于导航任务，输入处理方式相似，但每个对象、颜色和状态ID首先嵌入到 $d_{embed} = 8$ 中。而对于CTM，主干的输出与当前激活状态连接，而对于LSTM基线，输出则在与CTM相同数量的内部时钟周期后进行处理。演员和评论家头部实现为两层的多层感知机（MLP），每层包含64个具有ReLU激活的隐藏层。对于CTM，这些头部接收输出神经元的同步输入，而对于LSTM基线，它们接收LSTM在 T 内部时钟周期后的隐藏状态。密集配对用于选择同步的神经元。

与其他任务如图像分类（第3节），使用UNet风格的synapse模型不同，RL任务采用两层前馈synapse，每层由线性变换、GLU和LayerNorm组成。实验证明，这两层显著优于单层，特别是在导航任务中，单层synapse始终无法达到LSTM的平均回合长度。

T他为CartPole、Acrobot和MiniGrid 四个房间实验使用的模型超参数
c可以在表5到表7中找到。

Model	T	M	d_{model}	d_{input}	d_{hidden}	J_{out}	**Total Parameters**
CTM	1	10	128	128	4	16	175437
LSTM	1	–	118	128	–	–	175855
CTM	2	20	128	128	4	16	188237
LSTM	2	–	126	128	–	–	188863
CTM	5	50	128	128	4	16	226637
LSTM	5	–	148	128	–	–	227275

表 5 | CartPole 实验的模型超参数。

Model	T	M	d_{model}	d_{input}	d_{hidden}	J_{out}	**Total Parameters**
CTM	1	5	256	64	4	16	350094
LSTM	1	–	243	64	–	–	350118
CTM	2	10	256	64	4	16	362894
LSTM	2	–	249	64	–	–	364290
CTM	5	25	256	64	4	16	401294
LSTM	5	–	265	64	–	–	403490

表 6 | Acrobot 实验的模型超参数 源文本: .翻译文本: .

Model	T	M	d_{model}	d_{input}	d_{hidden}	J_{out}	Total Parameters
CTM	1	10	512	128	16	32	7802690
LSTM	1	–	294	128	–	–	7813692
CTM	2	20	512	128	16	32	7976770
LSTM	2	–	300	128	–	–	7979304

Table 7 | Model hyperparameters for the MiniGrid Four Rooms experiments.

I.3. Optimization details

The models were trained with Proximal Policy Optimization (Schulman et al., 2017) on single H100 Nvidia GPU. The same set of PPO hyperparameters were used for both the CTM and the LSTM baseline, and are shown in Table 8.

Hyperparameter	CartPole	Acrobot	MiniGrid Four Rooms
Learning Rate (LR)	1×10^{-3}	5×10^{-4}	1×10^{-4}
Total Environment Steps	10M	2M	300M
Rollout Length	50	100	50
Number of Environments	256	12	256
Max Environment Steps per Episode	200	500	300
Update Epochs	4	1	1
Minibatches	4	4	4
Discount Factor (γ)	0.99	0.99	0.99
GAE Lambda (λ)	0.95	0.95	0.95
Clip Coefficient	0.1	0.1	0.1
Entropy Coefficient	0.1	0.1	0.1
Value Function Coefficient	0.25	0.25	0.25
Value Function Clipping	No	No	No
Max Gradient Norm	0.5	0.5	0.5

Table 8 | PPO hyperparameters for each task.

J. UMAP

We used UMAP (McInnes et al., 2018) to build Figure 6. The purpose of UMAP in this case was to give each neuron in the ImageNet CTM a 2D location such that when their activities over time were visualized a meaningful pattern could be observed, should it exist. To this end, we considered the histories of post-activations for 200 different images as the high-dimensional inputs to UMAP ($200 \times T = 200 \times 5 = 1000$ dimensions). UMAP was then used to project this to a 2D space for visualization.

K. Recursive computation of the synchronization matrix

In Section 2.4 we defined the synchronization matrix at internal tick t as

$$\mathbf{S}^t = \mathbf{Z}^t (\mathbf{Z}^t)^\intercal, \qquad \mathbf{Z}^t \in \mathbb{R}^{D \times t}, \tag{13}$$

Model	T	M	d_{model}	d_{input}	d_{hidden}	J_{out}	Total Parameters
CTM	1	10	512	128	16	32	7802690
LSTM	1	–	294	128	–	–	7813692
CTM	2	20	512	128	16	32	7976770
LSTM	2	–	300	128	–	–	7979304

表 7 | MiniGrid 四个房间实验的模型超参数。

I.3. 最优化细节

T他使用了与Schulman等人（2017年）提出的proximal策略优化（Proximal Policy Optimization，{v*}）算法在单个Nvidia GPU。CTM和LSTM都使用了相同的PPO超参数。
baseline，并且如表8所示。

Hyperparameter	CartPole	Acrobot	MiniGrid Four Rooms
Learning Rate (LR)	1×10^{-3}	5×10^{-4}	1×10^{-4}
Total Environment Steps	10M	2M	300M
Rollout Length	50	100	50
Number of Environments	256	12	256
Max Environment Steps per Episode	200	500	300
Update Epochs	4	1	1
Minibatches	4	4	4
Discount Factor (γ)	0.99	0.99	0.99
GAE Lambda (λ)	0.95	0.95	0.95
Clip Coefficient	0.1	0.1	0.1
Entropy Coefficient	0.1	0.1	0.1
Value Function Coefficient	0.25	0.25	0.25
Value Function Clipping	No	No	No
Max Gradient Norm	0.5	0.5	0.5

表 8 | 每个任务的 PPO 超参数。

J. UMAP

我们使用了UMAP（McInnes等，2018）来生成图6。在这种情况下，UMAP的目的是为ImageNet CTM中的每个神经元分配一个2D位置，使得当可视化它们随时间的变化时，可以观察到有意义的模式（如果存在的话）。为此，我们考虑了200张不同图像的后激活历史作为UMAP的高维输入（$200 \times T = 200 \times 5 = 1000$维度）。然后使用UMAP将这些数据投影到2D空间进行可视化。

K. 递归计算同步矩阵

在第2.4节中，我们将内部时钟t处的同步矩阵定义为

$$\mathbf{S}^t = \mathbf{Z}^t (\mathbf{Z}^t)^\intercal, \qquad \mathbf{Z}^t \in \mathbb{R}^{D \times t}, \tag{13}$$

where the d–th row of $\mathbf{Z}^t$ stores the post–activation trace of neuron d up to tick t (cf. Eq. (4)). Because Eq. (13) recomputes all D^2 inner products from scratch at every tick, its time complexity is $O(D^2 t)$ over a roll-out of length t. Below we show that, with the exponentially–decaying rescaling of Eq. (10), the same quantity can be obtained from a pair of first–order recursions that require only $O(D_{\text{sub}})$ work per tick, where $D_{\text{sub}} \ll D$ is the number of subsampled neuron indices actually used for the output and action projections.

For notational clarity we first consider a single (i, j) neuron pair and omit the subsampling; the extension to a batch of pairs is immediate. Recall that the rescaled synchronization entry is defined as

$$S_{ij}^t = \frac{\sum_{\tau=1}^{t} \mathrm{e}^{-r_{ij}(t-\tau)}\, z_i^\tau\, z_j^\tau}{\sqrt{\sum_{\tau=1}^{t} \mathrm{e}^{-r_{ij}(t-\tau)}}}, \tag{14}$$

where $r_{ij} \geq 0$ is the learnable decay rate for the pair (i, j). Define the following auxiliary sequences

$$\alpha_{ij}^t := \sum_{\tau=1}^{t} \mathrm{e}^{-r_{ij}(t-\tau)}\, z_i^\tau\, z_j^\tau, \qquad \alpha_{ij}^1 = z_i^1 z_j^1, \tag{15}$$

$$\beta_{ij}^t := \sum_{\tau=1}^{t} \mathrm{e}^{-r_{ij}(t-\tau)}, \qquad \beta_{ij}^1 = 1. \tag{16}$$

Then $S_{ij}^t = \alpha_{ij}^t / \sqrt{\beta_{ij}^t}$ and both α_{ij}^t and β_{ij}^t obey simple first–order difference equations:

$$\alpha_{ij}^{t+1} = \mathrm{e}^{-r_{ij}}\, \alpha_{ij}^t \;+\; z_i^{t+1}\, z_j^{t+1}, \tag{17}$$

$$\beta_{ij}^{t+1} = \mathrm{e}^{-r_{ij}}\, \beta_{ij}^t + 1. \tag{18}$$

The rank–1 update in Eq. (17) makes it unnecessary to store the full activation history or to repeatedly form large outer products. During forward simulation we maintain α_{ij}^t and β_{ij}^t for each selected pair and update them in $O(1)$ time.

In practice we store $\{\alpha_{ij}^t, \beta_{ij}^t\}$ only for the two disjoint subsamples that form $\mathbf{S}_{\text{out}}^t$ and $\mathbf{S}_{\text{action}}^t$ (Section 2.4). Both memory footprint and compute overhead therefore scale linearly with the number of retained pairs, i.e. $O(D_{\text{sub}}) = O(D_{\text{out}} + D_{\text{action}})$ per tick.

其中，Z^t 的 d 行存储了神经元 d 在第 t (个 tick 的后激活轨迹，参见式 (4))。由于式 (13) 在每个 tick 都从头重新计算所有 D^2 个内积，因此其时间复杂度为 $O(D^2t)$ 在长度为 t 的展开中。下面我们将证明，通过式 (10) 的指数衰减重新缩放，同样的量可以从一对只需要 $O(D_{\text{sub}})$ 次工作的零阶递归中获得，其中 $D_{\text{sub}} \ll D$ 是实际用于输出和动作投影的子采样神经元索引的数量。

为了使符号更加清晰，我们首先考虑一个单一的 (i, j) 神经元对，并省略下采样；批量扩展是立竿见影的。回想一下，重新缩放的同步项定义为

$$S_{ij}^t = \frac{\sum_{\tau=1}^{t} \mathrm{e}^{-r_{ij}(t-\tau)}\, z_i^\tau\, z_j^\tau}{\sqrt{\sum_{\tau=1}^{t} \mathrm{e}^{-r_{ij}(t-\tau)}}}, \tag{14}$$

其中 $r_{ij} \geq 0$ 是可学习的衰减率，用于配对 (i, j)。定义以下辅助序列

$$\alpha_{ij}^t := \sum_{\tau=1}^{t} \mathrm{e}^{-r_{ij}(t-\tau)}\, z_i^\tau\, z_j^\tau, \qquad \alpha_{ij}^1 = z_i^1 z_j^1, \tag{15}$$

$$\beta_{ij}^t := \sum_{\tau=1}^{t} \mathrm{e}^{-r_{ij}(t-\tau)}, \qquad \beta_{ij}^1 = 1. \tag{16}$$

然后 $S_{ij}^t = \alpha_{ij}^t / \sqrt{\beta_{ij}^t}$ 和两个 α_{ij}^t 和 β_{ij}^t 遵循简单的二阶差分方程:

$$\alpha_{ij}^{t+1} = \mathrm{e}^{-r_{ij}}\, \alpha_{ij}^t \; + \; z_i^{t+1}\, z_j^{t+1}, \tag{17}$$

$$\beta_{ij}^{t+1} = \mathrm{e}^{-r_{ij}}\, \beta_{ij}^t + 1. \tag{18}$$

等式 (17) 中的秩-1 更新使得无需存储完整的激活历史或反复形成大型外积。在前向模拟过程中，我们为每个选定的配对维护 α_{ij}^t 和 β_{ij}^t，并在 $O(1)$ 时间内更新它们。

在实践中，我们只存储 $\{\alpha_{ij}^t, \beta_{ij}^t\}$ 两个不相交子样本，即 $\mathrm{S}_{\text{out}}^t$ 和 $\mathrm{S}_{\text{action}}^t$ (第 2.4 节) 中的。因此，内存占用和计算开销都与保留的配对数量成线性关系，即每 tick $O(D_{\text{sub}}) = O(D_{\text{out}} + D_{\text{action}})$。