Uncovering Neural Mechanisms of Mental Simulation With Programmed Attractor Networks

Daniel Calbick[1,2*], Jason Z. Kim[3], Hansem Sohn[4,5], Ilker Yildirim[1,2,6,7,8*]

1 Interdepartmental Neuroscience Program, Yale University
2 Wu Tsai Institute, Yale University
3 Department of Physics, Cornell University
4 Center for Neuroscience Imaging Research, Institute for Basic Science (IBS), Suwon, Republic of Korea
5 Department of Biomedical Engineering, Sungkyunkwan University (SKKU), Suwon, Republic of Korea
6 Department of Psychology, Yale University
7 Department of Statistics & Data Science, Yale University
8 Foundations of Data Science Institute, Yale University

*** daniel.calbick@yale.edu, ilker.yildirim@yale.edu**

Abstract

Predicting future states of a scene —a cognitive capacity often termed "mental simulation"— is essential for adaptive behavior, yet its neural mechanisms remain unclear. Cognitive theories suggest 'structure-preserving representations' (SPRs) [1] as the driver of mental simulations — functional mappings of the worldly causes that shape the way scenes unfold, akin to physics simulations in video game engines [2]. However, these cognitive theories, with their symbolic, program-like implementations, do not readily inform neural mechanisms. Here, we introduce a multilevel computational theory — *Programmed Attractor Networks (PANs)* — which formalizes mental simulation jointly across levels of analysis [3, 4] by bringing together structure-preserving, game-engine-style representations of cognitive science [5] with continuous attractor networks of neuroscience [6]. PANs are recurrent neural networks (RNNs) that are programmed (and not trained) with game-engine-style representations of physical scenes based on an algorithmic formulation of these representations as 'dynamical attractor manifolds'. We compare PANs to neural activity in the dorsomedial frontal cortex (DMFC) of macaques performing a naturalistic ball-interception task [7]. PANs recapitulate a striking, previously unexplained feature of DMFC activity and make a novel prediction that we confirm in neural data: The entire future trajectory of the ball becomes linearly decodeable in DMFC circuitry, and in PANs, right at the beginning of a trial. Finally, PANs also recapitulate the representational similarity of trial-level, momentary neural dynamics. Task-optimized statistical features in standard RNNs, as well as task-performant heuristics, differ qualitatively from DMFC neural activity. These results suggest that primate frontal circuits implement structure-preserving, physics-based representations of scenes through latent, dynamical attractor manifolds. PANs offer a general, integrative approach for uncovering how the knowledge of the world is implemented in neural mechanisms.

Main Text

Mental simulation — the ability to predict and reason about future states of a scene — is a fundamental cognitive capability for generating flexible behavior in complex, dynamic environments. This ability to "play forward" scenarios in our minds is thought to underlie many advanced cognitive functions such as intuitive physics, planning, and tool use [2, 8, 9]. Understanding how the brain implements these mental simulations at the circuit level remains a central challenge of neuroscience.

To address this knowledge gap, we leveraged a naturalistic ball-interception task, developed by Rajalingham et al. [7, 10]. In this task, rhesus macaques (*Macaca mulatta*) use a joystick to control a paddle to intercept a moving ball on a computer screen (Fig. 1a; much like the classical video game of Pong). During the late portion of its trajectory, the ball becomes occluded, requiring some prediction strategy for successful interception. The task design systematically varies the initial position and velocity of the ball across 79 unique conditions, yielding a range of trajectories with varying durations (range: 1450-3750 milliseconds). [A trial ends when the ball reaches the terminal (rightmost) side of the board, regardless of whether it is successfully intercepted with the paddle or not.] Neural activity was recorded from the dorsomedial frontal cortex (DMFC) of two macaques while they performed this task. Large-scale electrophysiological recordings yielded activity from 1,889 neurons. The DMFC populations contained information about the ball position regardless of whether it was occluded, making it a prime candidate for implementing mental simulation of physical scenes [7].

What computations underlie the observed DMFC neural activity? A long-standing tradition in neuroscience emphasizes the role of successful behavioral outputs in explaining how organisms represent the external world [11]. A crucial property of task-optimized representations, beyond efficiently supporting task performance, is that they can involve "shortcuts" and "simple tricks" [11, 12]. The advent of deep neural networks (DNNs) has facilitated computational explorations of task-optimized, statistical representations of the external world. Indeed, computational neuroscience has increasingly benefited from the use of task-optimized DNNs (e.g., [13–16]), which can be viewed as "shortcut learners", showing sensitivity to even small changes in how they are tested [17]. It is possible that DMFC also harbors such task-optimized statistical approximations of the board state.

A much different possibility for what computations might underlie DMFC activity comes from the field of cognitive science. Cognitive theories suggest 'structure-preserving representations' (SPRs) [1] as the engine of mental simulation [2]. SPRs correspond to the computational-level description of mental simulation as building and running forward a functional map (i.e., a homeomorphic map; see [1]) of the worldly causes that shape the way scenes unfold. For a physical scene, SPRs include a small number of physical properties sufficient to describe how objects move and react to external forces — much like the physics simulations in video game engines [2]. Functionally, SPRs are appealing: Having access to the worldly causes underlying a scene is behaviorally efficacious [1], supporting efficient learning and flexible generalization [19–21], and compositionality [22]. Despite this appeal and a growing support [23–29], these models, with their typically symbolic, program-like implementations, do not inform how biological neural systems could implement physics-based, structure-preserving representations of objects. Indeed, it remains unclear whether and how biological neural networks implement structure-preserving game-engine-style representations of the external world.

Which of these computational possibilities may better explain DMFC neural activity? To implement the first possibility of statistical and task-optimized representations, we use the DNNs from ref. [10]; crucially, these models are compelling candidates as they are shown to explain behavioral performance of both primates and humans in the current ball-interception task [10].

But how can we test the second possibility — that DMFC implements a structure-preserving game-engine style representation of physical scenes? This is challenging because typical implementations of SPRs involve symbolic, program-like components (e.g., [2, 25]), which do not readily inform how biological neural systems could implement simulation-based, structure-preserving representations of physical scenes. To overcome this limitation and so to test this possibility of DMFC implementing SPRs, we present a novel multilevel computational theory (where 'level' refers to Marr's three levels of analysis [4]) called *Programmed Attractor Networks* (PANs). At the computational level, this theory describes mental simulation as building and unfolding structure-preserving representations of physical scenes (and thus flexibly supporting many downstream adaptive behaviors) [2]. Here, such a symbolic representation encodes the properties and relations of the entities on the board (ball, paddle, walls), including the ball's position and how

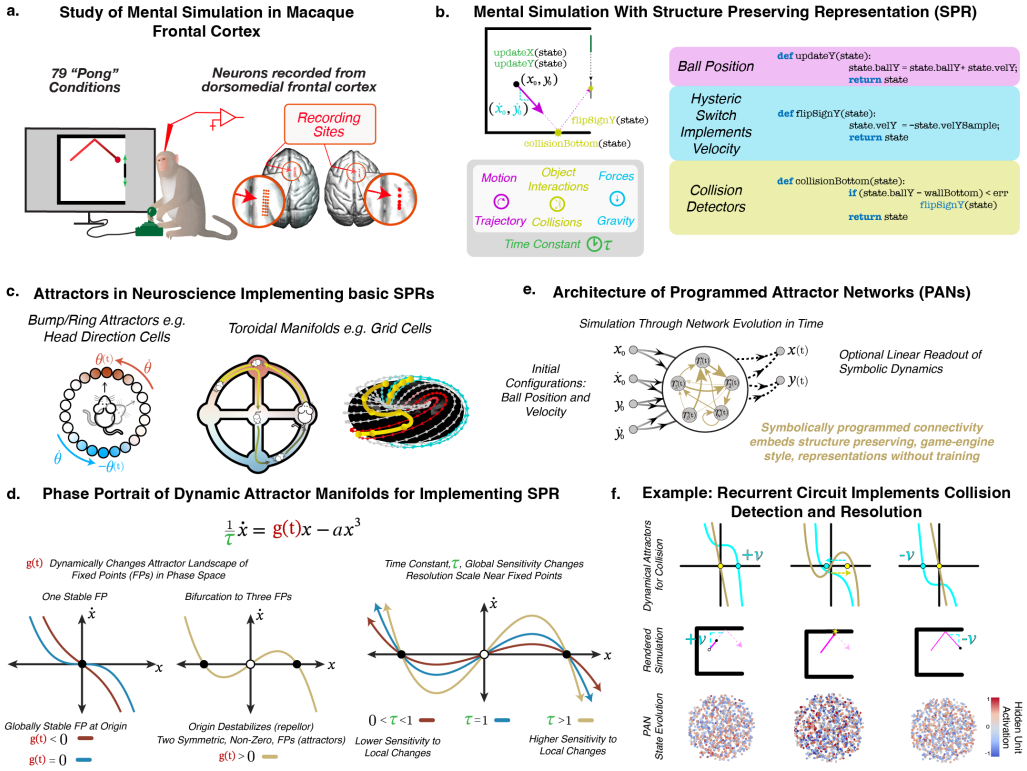


Figure 1. Overview of Programmed Attractor Network (PAN), a multilevel computational theory of the neural mechanisms of mental simulation. (a) Experimental design showing neural recordings from dorsomedial frontal cortex (DMFC) in two macaques performing a naturalistic ball interception task. (b) Computational-level description of a structure-preserving representation for mental simulation, including the entities of the ball, walls, and the paddle, their positions, velocities, and interactions. The symbolic description formally captures this physics-based structure-preserving representation in a computer program. (c) Examples of attractor systems in neuroscience implementing basic SPRs, including a ring attractor for heading direction and a torus for position in 2D space. (d) At the algorithmic level of PAN are dynamical attractors that provide a flexible computational primitive for expressing symbolic programs in panel b. Phase portrait of a controlled bifurcation with the equation $\frac{1}{\tau}\dot{x} = g(t)x - ax^3$, showing how stable fixed points (attractors) and unstable fixed points (repellers) can be modulated through the time-varying coefficient $g(t)$. These coefficients can couple different dynamical variables and implement branching logic, linear velocity dynamics, and other elements needed to represent the physical scene. (e) At the implementation level of PAN are RNNs that are programmed to embed the dynamical attractors in panel d. Specifically, we use a recent method [18] to program (and not train) the weights and connectivity of a reservoir computer with the algorithmic-level dynamical system (see text and Methods). Our analyses compare the activity of the programmed RNN to neural data. (f) Collision detection and resolution in PAN through dynamics and recurrent circuits that undergo phase transitions when the ball approaches a wall. Top: phase portrait showing state transitions at the algorithmic level (dynamical attractors; lines) and implementation level (RNN approximation of the dynamics; dots); Middle: rendering of the simulated trajectory; Bottom: Distributed computation illustrated by the individual values of the latent variables (color bar illustrates relative activation) at this time point during the simulations. The individual latent units were uniformly distributed around a disc for visualization to illustrate the high-dimensional RNN hidden state, whose fluctuations and recurrent connections support the lower-dimensional structure-preserving dynamical manifold.

it changes over time (linear dynamics), if-else branching for collision detection and force relations (Fig. 1b; see Methods and Supplementary Fig. 1 for a full description). What neural mechanisms could ground such structure-preserving representations of physical scenes in the brain?

An important clue comes from recent work in neuroscience: Low-dimensional, latent manifolds underlying neural populations are found to implement basic structure-preserving representations of the world [6] (Fig. 1c). That is, the geometry of these manifolds mirrors the entities and their relations of the represented system [30–33], such as ring attractor and toroidal geometry [6] for heading direction and position in 2D space, respectively. For example, in the heading direction circuitry of the fruit fly, neural activity traces a ring-shaped manifold corresponding to the circular nature of head orientation [32, 33]. It is plausible that the brain also encodes structure-preserving representations of physical scenes, but it remains unclear how to construct manifolds that compute these more sophisticated representations — beyond the relatively simple domains that have so far been explored.

This brings us to the algorithmic-level formulation of our theory: We suggest that in the brain, these game-engine style representations are encoded in latent 'dynamical attractor manifolds' (Fig. 1d). Unlike a standard manifold consisting of a fixed attractor configuration (e.g., a set of stable points that the dynamics of a system tends to), in dynamical attractor manifolds, we allow the stable points to change over time, by coupling multiple dynamical variables through the coefficients that determine their dynamics. This enables the expression of symbolic, program-like procedures in manifolds, including branching-dependent computations such as collision detection and resolution. To create these computational primitives, we leverage controlled bifurcations as introduced in a recent method from the physics of dynamical systems [18]. For example, by defining the standard cubic bifurcation $\frac{1}{\tau}\dot{x} = g(t)x - ax^3$, we obtain a flexible computational substrate where stable fixed points (attractors) and unstable fixed points (repellers) can be precisely modulated through time-varying parameters, denoted $g(t)$ (Fig. 1d, left). For collision detection and resolution, we exploit this property: When the ball is far from any boundary ($g(t) \ll 0$), the network acts as a pure velocity integrator with a single, globally stable fixed point at the origin. As the ball approaches a wall ($g(t) \geq 0$), the control parameter crosses a critical value and the system undergoes a pitchfork bifurcation: the origin loses stability and two new stable fixed points emerge (Fig. 1d, left). The state is rapidly drawn to one of these attractors, where a downstream hysteretic element flips the sign of the velocity, sending the ball back into the arena (see Methods). Additionally, the other coefficient in the above equation, the time constant τ, independently controls the accuracy ("sharpness") of collision detection and resolution by modulating the sensitivity of these dynamics to local changes (Fig. 1d, right). We express the structure-preserving representation in Fig. 1b by chaining additional dynamical attractors (see Methods and Supplementary Fig. 2). Much like a ring attractor captures the structure of heading direction, the resulting dynamical attractor manifold functionally maps the essential properties and relations in physical scenes.

Finally, at the implementation level, PAN realizes this dynamical attractor manifold in a recurrent neural network (RNN; Fig. 1e). To do so, we project the structure-preserving dynamical attractors onto the weights and connectivity of an RNN, specifically a tanh activated reservoir computer [34, 35], using the method of ref. [18]. This projection does not involve any data sampling or training on data (as commonly done in machine learning-style task-optimized deep neural networks). In brief, this method "decompiles" the RNN's hidden state and dynamics into an analytic basis of its inputs (using Taylor series expansion) and uses this basis to program the desired dynamical system (e.g., a cubic bifurcation parametrized by its coefficients, which are, crucially, also expressed analytically) into the adjacency matrix of the RNN (see Methods and Supplementary Information). The RNNs have much higher dimensionality than the underlying manifold (our main results use 1000 hidden units, versus the 12 dynamical attractors at the algorithmic level). By translating the symbolic and algorithmic language of differential equations into the adjacency matrix of the RNN, we shape the network's latent-space dynamics such that its attractor landscape recapitulates the computational logic of the dynamical attractor manifold (illustrated in a subset of the dynamics in Fig. 1f, see Supplementary Fig. 2 for a fuller and dynamic visualization).

This completes our description of the PAN model — a novel multilevel account of mental simulation, which embeds the cognitive hypothesis of a structure-preserving representation on one end and is fully testable in high-resolution neural data on the other. To compare PAN to neural data, we provide the model with the initial configuration of a given condition (for each of the 79 conditions the monkeys experienced), including the initial ball position and velocity. PAN unfolds this initial state via recurrent dynamics, without any additional external input. This approach mirrors the plausible process by which frontal circuits receive compressed sensory information from upstream areas to establish initial conditions, then perform forward simulations through latent dynamics. Following the approach in [7], we analyze the hidden state of PAN as well as the DMFC population activity in chunks of 50 milliseconds (average activation for each hidden unit of PAN and average spiking activity for each biological neuron, within the corresponding period of 50 milliseconds; Fig. 2a).

A striking feature of the neural data is the rapid encoding of the ball's final position where it is intercepted ("ball end-point") by 250 milliseconds after trial onset [7]. To replicate this result and to analyze models, we created our own neural decoding pipeline (Fig. 2a). For each trial, we constructed a neural state matrix at different time bins following trial start, then used a generalized linear model (GLM) [36] to decode the ball position at paddle interception. Fig. 2b-i shows our replication of Rajalingham et al.'s finding [7] that the ball end-point becomes quickly available in the DMFC population activity, right after the start of a trial, just 250 milliseconds in — i.e., long before the ball becomes occluded or the trial ends (in absolute terms, a duration of more than 1000 milliseconds and sometimes up to 3.5 seconds). Because 200-250 milliseconds roughly corresponds to the time it would take for signal transduction to this frontal region [37] (including information about the initial ball position and velocity), we can say that this rapid prediction occurs essentially immediately as the initial conditions of the trial become available to neurons in the DMFC. Does PAN recapitulate this feature of neural data?

Remarkably, PAN (Fig. 2b-ii) captures this rapid prediction ability with high fidelity. The correlation between the decoded ball positions from PAN's hidden states and the actual ball position at interception closely matches the pattern observed in the neural data, maintaining high correlation (Pearson's $r > 0.8$) from early time points through approximately 2250 milliseconds. In a finer-grained analysis, we separated the 79 conditions in the dataset into "zero-bounce" (direct trajectories without a wall collision) and "one-bounce" (trajectories with one wall collision) conditions (Fig. 2c). We found that the neural data shows robust ball end-point prediction for both condition types, with slightly better performance for zero-bounce trials. PAN also recapitulates this finer-grained pattern in the neural data.

Importantly, standard task-optimized RNNs fail to reproduce this key feature of the neural data (Fig. 2b-iii), a result we replicate and extend from Rajalingham et al. [7] (see Methods). These task-optimized RNNs acquire high-level statistical regularities in their training datasets needed to minimize the respective objectives under which they are trained. One group of RNNs, which we call "Next-time point RNNs", is trained on the combined objectives of predicting where the ball will be in the next time point and where to place the paddle for successful interception. Another group of RNNs, which we call "Paddle-only RNNs", is trained only on the objective of where to place the paddle. As plotted in Fig. 2b-iii, these RNNs show a gradual improvement in ball end-point prediction accuracy over time, failing to capture the early predictive capability observed in both the neural data and PAN. Moreover, these models qualitatively decouple from neural dynamics at the finer-grained analysis of "zero-bounce" and "one-bounce" conditions (Fig. 2c; see below for discussion). Crucially, these discrepancies are not due to the raw task performance of these task-optimized RNNs, which are previously shown to perform well at the ball interception task and align with monkey behavioral performance [10]. These results, that PAN recapitulates rapid prediction ability but the task-optimized RNNs do not, provide initial support that instead of task optimization, the computational-level objective of building and manipulating simulation-based structure-preserving representations, which are embedded within PAN, is a better explanation of neural dynamics.

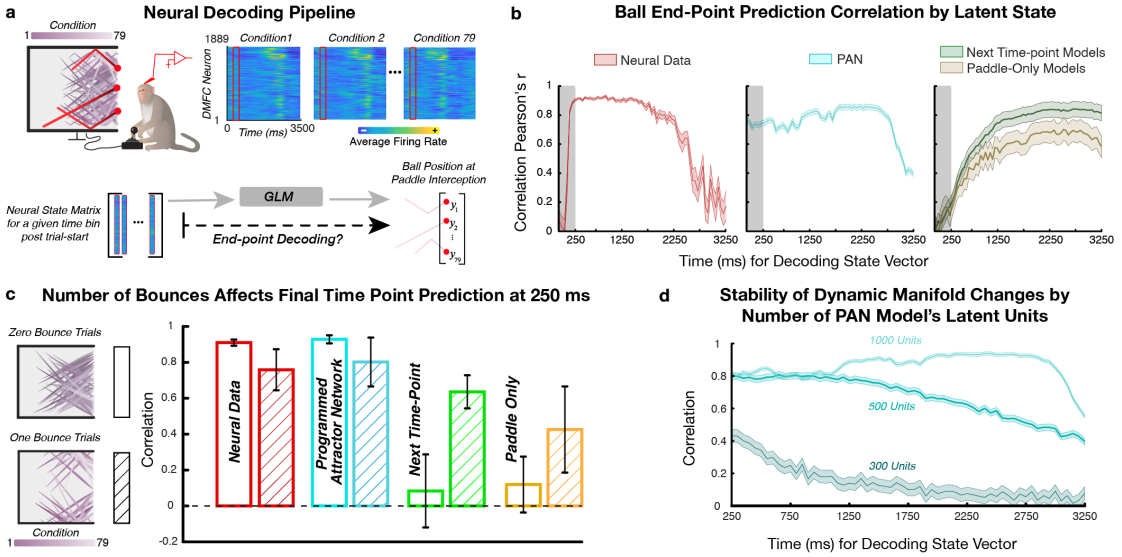


Figure 2. PAN explains rapid decodeability of ball end-point information in DMFC neural population. (a) Our decoding pipeline that is applied to both the neural data and models. This panel depicts the analysis of 1,889 DMFC neurons across 79 conditions in the dataset. For each time bin (from the start of the trial to the end), we used a generalized linear model (GLM) to decode the ball's position (x-axis) at paddle interception ("ball end-point") from the neural activity at that time bin. (b) Temporal evolution of decoding accuracy for the ball end-point from DMFC and model states. DMFC neural activity (left, red) shows early prediction capacity ($\sim$ 250ms after trial onset, essentially by the transduction time for neural activity to reach the frontal cortex), with high correlation (Pearson's $r > .8$) maintained until late in the trial (correlation drops beyond $\sim 2,250$ms due to the increasingly fewer number of conditions long-enough for training decoder). This pattern is closely matched by PAN (middle, cyan), while next-time point and paddle-only models (right, green/brown) show slowly improving prediction accuracy over time. (c) Effect of trajectory complexity on rapid (at 250 milliseconds) ball end-point prediction. DMFC populations show highly accurate ball end-point prediction for both zero-bounce and one-bounce trials, with slightly better performance on the less complex zero-bounce trials. Next time-point and paddle-only models show significantly poorer performance, especially for zero-bounce trials, indicating a "short-cut learning" strategy encouraged by task-optimization ([17]; see main text). In contrast, the PAN model, with its structure-preserving representations, recapitulates DMFC-like robust prediction for both condition types. (d) Stability of dynamical attractors is necessary for persistent ball end-point prediction over time in PAN. PANs with more units (1000 vs. 500 vs. 300) more accurately simulate scenes (Supplementary Fig. 3) and maintain stable predictions of ball end-point for more extended periods, indicating the importance of accurate dynamics for capturing DMFC-like computations.

But why do DMFC circuitry, and PAN, enable rapid ball end-point prediction, in a way that is sustained over time? Rajalingham et al. [7], based on the inability of the next-time point models to explain their data, concluded that the brain may be operating on two different "strategies": an offline prediction that occurs early in the trial (with an unspecified mechanism), and online simulation-based next-time point predictions in the rest of the trial. PAN suggests a dramatically different possibility: The latent, dynamical attractor manifold, once configured by the perceived initial configurations of a trial, renders the ball end-point a linear projection along the surface of this manifold. As an initial test of this possibility, we varied the accuracy of the dynamical attractor manifold by reducing the number of hidden units in PANs. Bigger networks more accurately simulate the board (Supplementary Fig. 3), suggesting that sufficient dimensionality is

required to approximate the physical scene. Crucially, we found that these bigger networks also maintained stable predictions of ball end-point for longer periods (Fig. 2d), similar to the DMFC population. This result suggests a neural mechanism of mental simulation based on latent dynamical attractor manifolds of structure-preserving, game-engine-style representations.

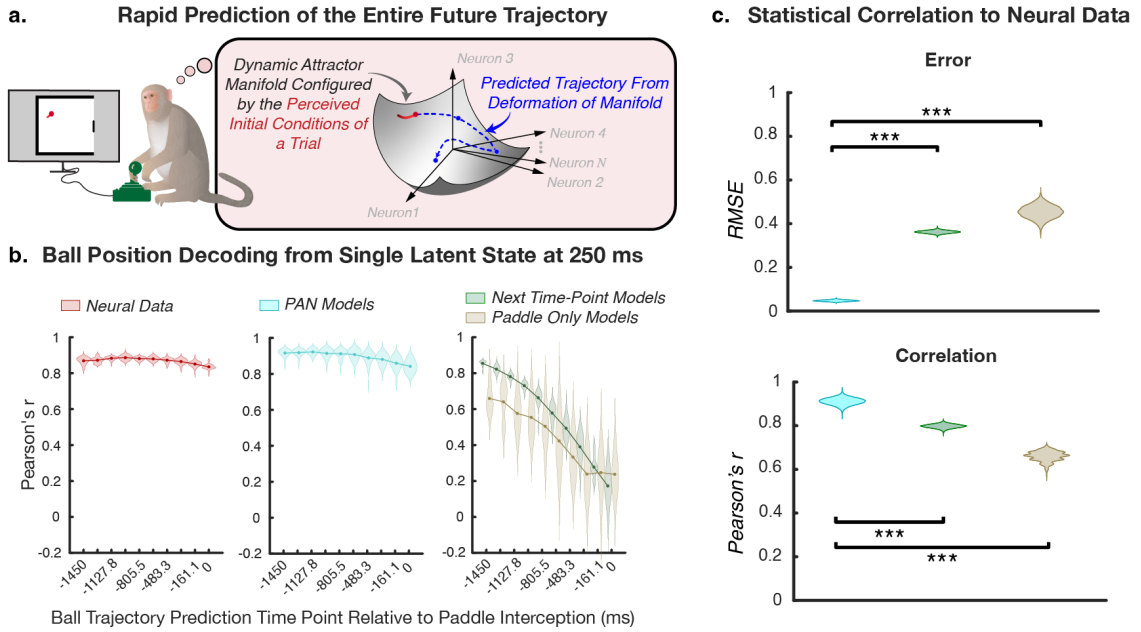


Figure 3. Confirming a striking prediction of PAN: at the trial start, the complete future trajectory of the ball, not just the end-point, is decodeable in neural activity. (a) A schematic of the logic of PAN's prediction. Once the latent dynamical attractor manifold within the high-dimensional neural activity is configured with the perceived initial conditions of a trial (i.e., initial ball position and velocity), different points along the future trajectory of the ball become just as linearly decodable as the ball end-point, as a lawful trajectory along the surface of this manifold. (b) Decoding accuracy for the ball's position across multiple points along its future trajectory (analyzed from -1450ms relative to paddle interception), using neural and model state vectors only at 250 milliseconds after trial onset. PAN's structure-preserving dynamical attractor manifold renders the entire future trajectory of the ball linearly decodeable (middle, cyan). Remarkably, we confirm this pattern in neural data. Early neural activity in DMFC, only at 250 milliseconds, supports highly accurate linear decoding (Pearson's $r > 0.8$) of the entire future trajectory of the ball (left, red). Despite their high task (intercepting the ball with the paddle) performance, next-time point and paddle-only models (right, green/brown) show decreasing accuracy for temporally farther trajectory points, with particularly poor performance for the most distant time points. (c) Statistical comparison of the entire future trajectory decoding performance of models. Violin plots show the distribution of prediction errors (left) and correlation values (right) when comparing models to neural data. PAN achieves significantly lower error rates and higher correlation, relative to both next-time point and paddle-only models (***: $p < .001$).

This result also leads us to a novel prediction of the PAN model: Because the ball end-point is not coded in any special way in the structure-preserving representation of the physical scene, we predict that at the start of the trial, not only the end-point but also the *entire future trajectory* of the ball will be linearly decodeable, both in DMFC and PAN. The logic of this prediction is illustrated in Fig. 3a — a manifold configured by the perceived initial conditions of a trial renders future states available. Remarkably, we confirm this prediction (Fig. 3b) in both the neural data and PAN. When decoding from neural states at 250 milliseconds after trial onset, we can

reconstruct the ball's position at any point along its future trajectory with high accuracy (Pearson's $r > 0.8$; Fig. 3a-i). PAN replicates this capability, maintaining high correlation throughout the trajectory prediction window (Fig. 3a-ii). Crucially, these results need not be as we found: Indeed, models that show high task performance (i.e., accurately intercepting the ball, in ways aligned with monkey performance [10]), including the next time-point and paddle-only models, do not recapitulate such rapid full trajectory prediction. Instead, these models show decreasing prediction accuracy for more distant future time points, inconsistent with the neural data and PAN (Fig. 3a-iii). Statistical analysis (Fig. 3b) confirms that PAN achieves significantly higher correlation and lower error compared to alternative models. These results demonstrate that in PAN, the initial conditions place the dynamical attractor manifold in a configuration where the future is a "linear decoder away". This indicates a neural mechanism of mental simulation in DMFC based on dynamical attractor manifolds of structure-preserving, game-engine-style representations.

The simplicity of the board in the current interception task allows for an alternative heuristic-based, non-simulation strategy for predicting the ball's end-point: linearly mapping the initial ball position and velocity to its final position (the "Linear Map" heuristic; Supplementary Fig. 4). Does the DMFC neural activity employ this specialized heuristic strategy, instead of the structure-preserving nonlinear dynamical attractors that PANs stipulate? To answer this, we analyzed a divergent prediction made by the Linear Map heuristic and PANs: the generalization performance of the end-point decoder across conditions with 0-bounce versus 1-bounce. Linear Map yields a time-invariant, nearly perfect generalization of the end-point decoder across the bounce conditions. In contrast, PAN, due to its bifurcating dynamics, yields a time-dependent and less performant generalization of the end-point decoder across the bounce conditions (Supplementary Fig. 4). When we apply this cross-bounce end-point decoder analysis to the DMFC activity, we find that it more closely aligns with PAN, showing time-dependence of cross-bounce generalization (Supplementary Fig. 4). That DMFC qualitatively decouples from the Linear Map heuristic provides important evidence for our dynamical attractors account. Despite the availability of a simpler solution that generalizes perfectly (i.e., the Linear Map heuristic), DMFC implements condition-specific nonlinear dynamics (e.g., bifurcating attractors). We suggest that this result reflects DMFC's evolution for diverse 3D physical scenes, not just 2D ball tracking on a simple board. Indeed, both DMFC and PANs match the end-point decoding performance of the Linear Map heuristic within each bounce type (both achieve ~ 0.8 correlation; see Fig. 2c for PAN results), while implementing rich condition-specific dynamics within the population. This computational strategy may explain the involvement of frontal circuitry across diverse sensorimotor behaviors requiring physics prediction, including reaching movements [38], pursuit and evasion [39], and object manipulation [40]. Together, this suggests the brain prioritizes structure-preserving representations over computational shortcuts, even when shortcuts or simple learned heuristics would suffice.

Finally, we also tested the ability of PAN and alternative models to explain the pattern similarities of neural dynamics across pairs of moments and trials using representational similarity analysis (RSA; see Methods) [41]. Relative to the decoding-based analysis we have so far focused on, RSA imposes a qualitatively different test of candidate models for explaining the neural data. For a given data source (either a model or neural data), we built a representational similarity matrix where each cell is the correlation between the activity at time-point t_i in condition k_i and the activity at time-point t_j in condition k_j (Fig. 4a). We found that the similarity matrix of PAN achieves significantly and substantially higher correlations with the neural similarity matrix ($r = 0.59$), compared to the similarity matrices of the next-time-point ($r = .35, p < .001$) and paddle-only ($r = .08, p < .001$) models (Fig. 4b). We then used partial correlation analysis to ask the extent to which the PAN and next-time-point models explain non-overlapping variance in the neural similarity matrix (Fig. 4c; see Methods). We found that PAN explained a substantial amount of variance even after residualizing both the next-time-point and paddle-only models; but this was not the case for the next-time-point models which had nearly no variance to explain after residualizing the PAN and paddle-only models (Fig. 4d). The differences in the residualized variances explained by PAN versus next-time-point models were statistically significant ($p < .001$,

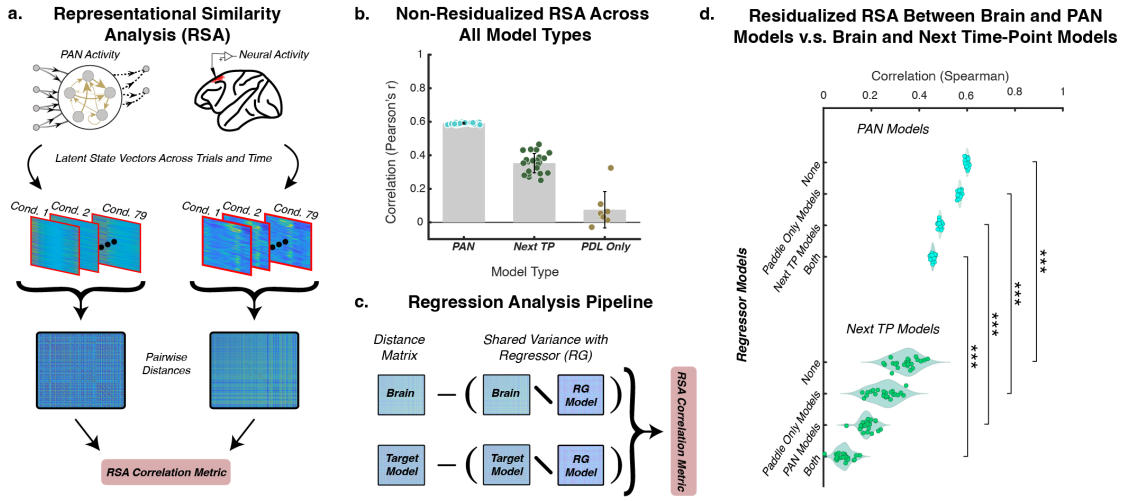


Figure 4. PAN captures the similarity structure of DMFC neural dynamics across moments and conditions. (a) We performed a representational similarity analysis comparing DMFC neural similarity matrix to the similarity matrices of each model. (b) PAN achieves a significantly higher correlation (Spearman's ρ) than the alternatives with lesser structure-preserving representations. (c) A schematic of the partial regression pipeline to determine and compare unique variance explained by different model types. (d) Correlation (Spearman's ρ) between model and neural representational similarity matrices is shown. The plot is organized by model type (top: PAN; bottom: next-time point) and residualization condition (controlling for variance explained by different model types). PAN shows significantly higher correlations with neural data than next-time point models across all residualization settings (***: $p < .001$). Critically, when residualizing PAN from next-time point models, correlation drops substantially, while PAN maintains a high correlation even after residualizing out all alternative models (next-time point and paddle-only models). This asymmetry indicates that structure-preserving RNNs capture fundamental aspects of neural computation not present in alternative models.

Fig. 4d). Moreover, we also evaluated an "Oracle" covariate (following [42]) that represented each time point for each condition using the ground-truth position and velocity of the ball (the same information that the Linear Map heuristic receives as input). PAN not only statistically significantly and substantially outperformed this covariate ($r = .59$ for PAN versus $r = .32$ for Oracle covariate), but also subsumed all of the variance it could explain (Supplementary Fig. 5). These results strongly suggest a neural mechanism of mental simulation through latent, structure-preserving dynamical attractors underlying DMFC activity, rather than heuristics or task-optimized statistical representations.

Distinctively, the present work synthesizes two prominent modeling paradigms of cognitive science and neuroscience, which have so far been developed largely independently of each other: Structure-preserving representations of cognitive science [1] based on symbolic, program-like formalizations [43] and the continuous attractor networks of neuroscience based on low-dimensional manifolds [6]. Unlike typical cognitive models that focus on computational-level explanations and behavioral data, PAN penetrates through levels of analysis and targets neural mechanisms [3]. And instead of the common neuroscientific approach of analyzing high-dimensional neural data through dimensionality-reduction techniques to visualize low-dimensional manifolds (e.g., [44]), our approach provides a computationally constructive handle on these manifolds. By doing so, this work suggests a neural mechanism of mental simulation in macaque DMFC, as latent dynamical attractor manifolds computing structure-preserving, physics-based representations of scenes. Just as the brain is attuned to statistical regularities in sensory inputs, our work shows that it is also attuned

to the laws of physics that govern the world around us.

Afforded by the experimental design of [7], we focused on mental simulation without addressing the problems of perception and action. Future work should extend PAN to scenarios with more complex mental simulation demands (e.g., [45]) as well as to scenarios where perception, prediction, and planning are more interdependent (e.g., the "prey pursuit" task of [39]). More generally, PAN's dynamical attractor manifolds — as a computational abstraction interfacing high-dimensional neural activity and symbolic programs — provide a general formalism that can inform the many multilevel computational challenges of neuroscience, from grounding higher-order cognitive representations in neural mechanisms to linking function and circuitry [46–50].

Critically, there was no a priori reason for why DMFC should be better explained by PAN, relative to the otherwise task-performant DNNs or other heuristic strategies. In fact, it is plausible that there is another brain region, e.g., early visual regions such as area MT, V1 or the primary motor cortex, where these alternatives provide a better account than PAN. We suggest that PAN better explains DMFC circuitry because both compute SPRs, and the task optimization of the DNNs come at the expense of structure preservation. Specifically, our analysis indicates a form of "shortcut learning" [17], wherein the task-optimized models or heuristic strategies take shortcutswithout involving or recovering a fully structure-preserving representation of the board. For the task-optimized DNNs, this shortcut strategy appears more pronounced in trials that afford it more (the easier zero-bounce trials, relative to the harder one-bounce trials). However, we hasten to acknowledge the speculative possibility that there may be choices of training objectives, architectures, and training datasets, besides those that we evaluated, that will lead to task-optimized neural networks with DMFC-like neural dynamics [47, 51, 52]. So far, these explorations have limited ability to match neural dynamics [42]. Our commitment is that such a DNN will converge to a dynamics that is homeomorphic to our dynamical attractor manifold (e.g., not necessarily the identical set of 12 equations we used in our implementation, but a representation in a structure-preserving relationship to it).

The present work has implications for uncovering the computational foundations of biological intelligence. In biology, prey species are often born with the ability to evade predators and seek safety in a complex, dynamically changing world, pointing at the possibility of sophisticated precocial physical prediction with little or no opportunity for experiential learning. Moreover, the biological networks controlling these behaviors support rapid and flexible experiential learning in survival-critical behaviors [53, 54]. Similarly, despite the helplessness of human infants, developmental psychology suggests a sophisticated starting point for human cognition [55]. By programming physics-based representations into neural dynamics, we achieve what biology demonstrates: efficient, robust, and interpretable intelligence that begins with, rather than learns, the fundamental rules governing our world.

Methods

Programmed Attractor Networks (PANs)

Here we provide the details of the computational, algorithmic, and implementation levels of PAN.

Computational-Level Description: Structure-preserving, game-engine-style representations for mental simulation

At the computational level, mental simulation can be described as building and manipulating physics-based structure-preserving representations of the physical world [2], flexibly supporting downstream adaptive behaviors. Following Battaglia et al. [2], who programmed an off-the-shelf physics engine to make a working cognitive hypothesis of representing and "playing forward" physical scenes, we provide a similar structure-preserving game-engine style representation of the interception task in Supplementary Fig. 1. This symbolic program describes the entities (objects,

walls, paddle) and their dynamics and interactions, using a high-level object-oriented programming language. Battaglia et al. used this kind of program to explain human behavioral performance. In contrast, here our goal is to explain neural dynamics and distributed codes in population activity by which these structure-preserving representations may be implemented. PAN accomplishes this goal by first expressing these symbolic formulation algorithmically using dynamic attractor manifolds (next section), and then implementing this dynamical system in an RNN (the section after).

Algorithmic-Level Realization: Dynamic Attractor Manifolds

Recent work in neuroscience provides evidence that basic structure-preserving representations, such as one's heading direction or 2D position in space, are encoded in low-dimensional, latent manifolds underlying high-dimensional neural activity. These manifolds contain attractor dynamics (stable and unstable points of attraction and repulsion) that functionally map entities and their relations in the world. However, to realize our computational-level hypothesis, we need to express significantly more sophisticated computations in these dynamics.

To do so, we build on Kim & Bassett [18] to formulate a 12-dimensional dynamical system, whose outputs are denoted as $\{z_1, z_2, \ldots, z_{12}\}$. These dynamical attractors of this system — i.e., nullclines in phase space that change with time t — correspond to an SPR of the board and its dynamics, including position updates, as well as collision detection and resolution. Because these equations are coupled, each attractor affects the others, together forming the dynamical attractor manifold computing the structure-preserving, game-engine-style representation of our environment. The 12 dynamical variables implement four main computational blocks (corresponding to the four code blocks in Supplementary Fig. 1) based on two forms of third-order polynomial logic.

We first describe these polynomials.

- **Pitchfork bifurcation** where $f(t)$ controls whether we have a single stable attractor at the origin, ($f(t) \leq 0$), or two symmetric non-zero stable fixed points at $(\pm\sqrt{f(t)}, 0)$ and one unstable attractor at the origin, ($f(t) > 0$), (textbook example, [56])

$$\frac{1}{\tau}\dot{z} = f(t)z - z^3$$

- **Polynomial-Shift Operator** The second cubic simply slides the attracting point left or right without changing its stability. The time-varying term $f(t)$ raises or lowers the cubic, while the constants α and β tune the slope so that $|z|$ remains ≤ 1. Keeping the state in this range guarantees that the activity vector of our recurrent network never leaves the ball-park set by the weight matrix's spectral radius (roughly, the largest eigenvalue magnitude) [57], preventing runaway excitation

$$\frac{1}{\tau}\dot{z} = -\alpha z^3 + \beta z + f(t)$$

In these equations, the coefficient τ controls the sensitivity of the dynamics encoded in these polynomials to local changes (as well as initial conditions). Specifically, when $|\tau| \gg 0$, then the derivatives on either side of the fixed points are much stronger (as illustrated in main text Fig. 1d, right), making slight changes in the system more sensitive to changes. (This time constant is multiplicative to the global time constant of our RNNs, represented by γ in our update equation in the next section.)

The four computational blocks of our dynamical attractor manifold are as follows.

1. Constant registers (velocity components)

   To make each trial's initial velocity available to the rest of the network, we reserve two dedicated state variables, z_1 and z_2, whose dynamics are purely integrative registers:

$$|v_x| \mapsto \dot{z}_1 = 0 \tag{$\dot{z}_1$}$$
$$|v_y| \mapsto \dot{z}_2 = 0 \tag{$\dot{z}_2$}$$

Because $\dot{z}_1 = \dot{z}_2 = 0$, these variables act as constants during the simulation, giving the PAN a read-only handle on (v_x, v_y) while re-using the same recurrent weights across all trials.

2. Velocity Logic

$$\dot{x} \mapsto \dot{z}_3 = z_3 + (z_5)z_1 \tag{$\dot{z}_3$}$$
$$\dot{y} \mapsto \dot{z}_4 = z_4 + (z_7)z_2 \tag{$\dot{z}_4$}$$

3. Hysteretic Switch for Nonlinear Reflections.

   Each state variable takes values in $[-x_c, x_c]$, with

$$-x_c \equiv \text{“low” (0)}, \qquad +x_c \equiv \text{“high” (1)}.$$

   The dynamic logic couples two recurrent variables, the wall-collision variable (z_{11}) and the opposite hysteretic variable $z_5 \leftrightarrow z_6$ and $z_7 \leftrightarrow z_8$

$$f(y_i, y_j) = \frac{(y_i - x_c)(y_j - x_c)}{2x_c} - x_c$$

   implements a Boolean NAND gate:

$$(y_i, y_j) \longmapsto f = \begin{cases} -x_c & \text{if } y_i = +x_c \text{ or } y_j = +x_c, \\ +x_c & \text{if } y_i = y_j = -x_c. \end{cases}$$

   Because the output itself sits at a stable fixed point ($\pm x_c$), it retains its state against small perturbations — i.e. it forms a hysteretic memory element. Cross-coupling two such units yields a bistable attractor manifold that we use as the circuit's nonlinear reflection (bounce) switch.

$$\text{sign}(v_x) \mapsto \tfrac{1}{20}\dot{z}_5 = -\alpha z_5^3 + \beta z_5 + f(z_{11}, z_6, t) \tag{$\dot{z}_5$}$$

$$-\text{sign}(v_x) \mapsto \tfrac{1}{20}\dot{z}_6 = -\alpha z_6^3 + \beta z_6 + f(z_{12}, z_5, t) \tag{$\dot{z}_6$}$$

$$\text{sign}(v_y) \mapsto \tfrac{1}{20}\dot{z}_7 = -\alpha z_7^3 + \beta z_7 + f(z_9, z_8, t) \tag{$\dot{z}_7$}$$

$$-\text{sign}(v_y) \mapsto \tfrac{1}{20}\dot{z}_8 = -\alpha z_8^3 + \beta z_8 + f(z_{10}, z_7, t) \tag{$\dot{z}_8$}$$

4. Collision Detection, where the extent of the board is defined by its height (h) and its width (w), p_x and p_y are the paddle's horizontal position and vertical position respectively, and ϵ and σ define the sensitivity/resolution of the collision detector relative to the ball's distance to the walls/paddle.

$$\text{Top Collision } [0,1] \mapsto \tfrac{1}{1000}\dot{z}_9 \quad = -(z_9)^3 + (z_4 + (h-\epsilon))z_{10} \tag{$\dot{z}_9$}$$

$$\text{Bottom Collision } [0,1] \mapsto \tfrac{1}{1000}\dot{z}_{10} \quad = -(z_{10})^3 - (z_4 + (-h-\epsilon))z_{10} \tag{$\dot{z}_{10}$}$$

$$\text{Left Collision } [0,1] \mapsto \tfrac{1}{1000}\dot{z}_{11} \quad = -(z_{11})^3 - (z_3 + (-w-\epsilon))z_{10} \tag{$\dot{z}_{11}$}$$

$$\text{Right/Paddle Collision } [0,1] \mapsto \tfrac{1}{2000}\dot{z}_{12} \quad = -(z_{12})^3 + ([(z_3 - p_x)^2 + (z_4 - p_y)^2] - (w-\sigma))z_{12} \tag{$\dot{z}_{12}$}$$

Implementation Level: Programming a Reservoir Computer

In contrast to the traditional task-optimization approach, we constructed a separate class of RNNs by analytically embedding the dynamical attractor manifold defined above (the twelve differential equations encoding ball motion, wall collision logic, and velocity sign changes) into a given network's recurrent connectivity. We do so by following the method of Kim & Bassett [18].

The basic steps of this method are as follows.

1. **Start from the standard continuous-time echo-state equation**

$$\frac{1}{\gamma}\dot{\mathbf{r}} = -\mathbf{r} + \tanh(A\mathbf{r} + B\mathbf{x} + \mathbf{d}), \tag{1}$$

with tanh nonlinearity. We draw B i.i.d. from $\mathcal{U}[-0.5, 0.5]$; A is initially set to for the open-loop solution which will be "programmed" later. (note: there are multiple notational conventions for expressing the time constant of a differential equation: $\tau\dot{x} = f(x)$ or $\frac{1}{\gamma}\dot{x} = f(x)$. Here we use $\tau = \frac{1}{\gamma}$, in line with the convention of Kim & Bassett. [18])

2. **Solve hidden state as a function of inputs.** Because the leakage term linearises Eq. 1 around a randomly chosen operating point, r^*, we can solve for our bias term given r^*

$$d = \operatorname{atanh}(r^*) + Bx$$

and the solution to the echo-state equation can be expressed as a smooth map

$$\mathbf{r}(t) = h\big(\mathbf{x}, \dot{\mathbf{x}}, \ddot{\mathbf{x}}, \dots\big). \tag{2}$$

3. **Symbolic expansion.** Expand Eq. 2 to k^{th} order via a multivariate Taylor series, obtaining a "design matrix"

$$\mathcal{R} = \mathscr{T}_k[h] \in \mathbb{R}^{N \times k}. \tag{3}$$

Each column is a monomial basis function of the inputs, their time derivatives, and the multivariate interaction terms up to order k.

4. **Program the desired vector field.** Let $\mathcal{O} \in \mathbb{R}^{m \times k}$ hold the same monomials but evaluated on the *target* dynamics $\dot{\mathbf{z}} = f(\mathbf{z})$. Compile the programmed readout

$$W = \arg\min_W \|W\mathcal{R} - \mathcal{O}\|_F, \tag{4}$$

where $\|\cdot\|_F$ denotes the Frobenius/L_2 norm. This yields $W\mathcal{R} \simeq \mathcal{O}$ and hence

$$W\Big(\mathbf{r} + \tfrac{1}{\tau}\dot{\mathbf{r}}\Big) \;\approx\; \mathbf{z} + \tfrac{1}{\tau}f(\mathbf{z}). \tag{5}$$

5. **Load initial conditions.** Inject a given trial's starting state $(\mathbf{z}_0, \dot{\mathbf{z}}_0)$ through the input channel to set the reservoir at its *conditional operating point*, r_i^*:

$$\mathbf{r}_i^* = \tanh\big(B\mathbf{z}_0 + \mathbf{d}\big). \tag{6}$$

This latent vector, z_0, contains the displacement of r^* in our N-dimensional space such that the networks time evolution from this point indexes a unique board configuration (initial condition for our update equation) (ball position & velocity).

6. **Close the loop.** Split $B\mathbf{x}$ into a recurrent part $\hat{B}\hat{\mathbf{x}}$ and an exogenous part $\bar{B}\bar{\mathbf{x}}$, then substitute $\hat{\mathbf{x}} = W\mathbf{r}$:

$$\frac{1}{\gamma}\dot{\mathbf{r}} = -\mathbf{r} + \tanh\big((\hat{B}W)\mathbf{r} + \bar{B}\bar{\mathbf{x}} + \mathbf{d}\big). \tag{7}$$

This yields the effective adjacency $A^\star = \hat{B}W$. With the loop closed, we numerically integrate Eq. 7 (Runge–Kutta 45) from $\mathbf{r}_i^*$ to simulate the board dynamics; the observable state can be optionally read out via $W\mathbf{r}(t)$.

Full derivations and hyperparameter choices appear in Supplementary Material "Details of Programming RNNs"; see [18] for the original method.

Applying PAN to Experimental Conditions

Once we have our programmed recurrent neural network, we can "load in" the initial conditions and evolve our network over time. As the network evolves its distributed computation, across the N latent neurons, together computes the symbolic dynamical attractors that constitute our physics simulator for the board.

We initialize PANs with the initial conditions used within the monkey experiments. The initial conditions contain ball position and velocity for each trial, which PAN evolves autonomously without external inputs. After loading initial conditions, the network was allowed to run for the duration of the trial, producing hidden-state trajectories that captured the same fundamental physics constraints present in the real task. This approach allowed a direct mechanistic test of whether physics-based SPRs better match DMFC recordings than task-optimized statistical representations or heuristics.

Traditional Task-Optimized Deep Neural Networks

Rajalingham et al. [58] trained a large ensemble of standard ML-style recurrent neural networks (RNNs) on the same interception task, using standard supervised learning protocols. These networks — here referred to as "task-optimized RNNs" — took the ball's visual inputs and were optimized to predict and/or control the paddle's position in order to intercept the ball. Their training aimed solely to minimize their training objective, not to fit any neural data.

The next-time-point models were trained to both the ball position in the next time step and to control the paddle (where the paddle should be for successful ball-interception at the end of the trial). They considered three variants of the next-point models depending on the specifics of the loss and architecture for predicting the ball's position in the next time step. Here, we analyzed all these three variants together as they did not differ from each qualitatively or quantitatively.

The paddle-only models were trained only the latter objective — the ball end-point.

They considered different hyperparameter choices [e.g., number of units, RNN circuit type (GRU, LSTM), input representation, etc.]. Here we report the best performing variants (number of hidden units=40; RNN circuit type=LSTM or GRU, input representation=motion filters; pixel input or Gabor-filtered input). We used these task-optimized RNNs "as is", extracting their hidden states on each condition for comparison with DMFC recordings and PAN.

Stimuli and Neural Data

Two adult macaque monkeys (*Macaca mulatta*), one male (Monkey P) and one female (Monkey M), participated in this study. Animals performed a naturalistic ball interception task developed by Rajalignham et al. [7, 10]. In this task, each trial began with the ball in a random initial position and velocity in a two-dimensional arena. The ball traveled rightward at a constant speed, with zero or one bounce off the horizontal walls. Crucially, the ball was rendered only for the early portion of each trial; its trajectory then became occluded by a virtual "occluder" before reaching the far right side. The monkeys controlled a paddle positioned at the right edge via a one-degree-of-freedom joystick, attempting to intercept the ball upon its (unseen) arrival. Each trial ended when the ball either made contact with the paddle or exited the right boundary of the display. Inter-trial intervals were 750 milliseconds. Monkeys were rewarded with a juice drop when they successfully intercepted the ball. They were free to move their eyes during the occluded period and routinely shifted gaze in ways consistent with anticipating future ball positions.

Neural signals were recorded from DMFC using high-channel-count silicon probes (Neuropixels) in one animal (Monkey M) and linear probes (Plexon V-probes) in the other (Monkey P). Recording sites spanned an 8 mm × 3 mm grid in Monkey P (24 distinct locations, each sampled in two sessions) and a narrower grid in Monkey M (6 locations, each sampled in one session). Neurons were neither preselected nor excluded based on their response properties, and recording sites were not chosen based on putative task selectivity. Spikes were sorted with an automated algorithm (Kilosort 3.0) and subjected to quality checks that removed unstable or noisy units. Per-trial, per-neuron spike counts were binned in 50-milliseconds intervals. Only units with significant split-half reliability ($p < 0.01$) were retained, resulting in a final dataset of 1,889 reliably recorded neurons across both monkeys. Trials were organized into 79 unique stimulus conditions.

Neural Data Analysis and Model-Data Comparisons

Trajectory Decoding from State Vectors

To test the future decoding capabilities of latent state representations of DMFC and different models, we used generalized linear model (GLM) [36] fits to decode the positions of the ball from the latent state-vectors across time. Neural population responses, task-optimized RNN hidden states, and PAN models' states were all compared via two complementary analyses: ball end-point decoding (Fig. 2) and full trajectory decoding (Fig. 3). All GLM results reported are cross-validated using nested 5-fold cross-validation with 4-fold inner cross-validation for hyperparameter selection.

For ball end-point decoding, we regressed each population's activity at each time point onto the ball's actual position at the time it would leave the board (where it would be intercepted by the paddle). For full trajectory decoding, we regressed each population's activity at 250 milliseconds onto the ball's actual position along its trajectory (including endpoint). The x-axis in Figure 3b represents time relative to paddle interception (0 ms). Since trials varied in duration due to different ball velocities and starting positions (ranging from 29 to 77 time bins at 50 ms resolution), we aligned all trials to their endpoints. To maximize the amount of training/testing data available across all trials, we could only decode positions going back 1450 ms from interception—corresponding to the duration of the shortest trial (29 time bins × 50 ms). This ensures that neural data from the 250 ms time point could be used to decode ball positions at all time points shown for every trial in our dataset.

Similarity Analysis

Representational similarity analysis (RSA) evaluated the geometry of each latent space by computing pairwise Euclidean distances among the (79 conditions x 71 time bins), creating representational dissimilarity matrices (RDMs) for each model and the neural data. We then computed Spearman correlations between model and neural RDMs to assess representational

similarity. To determine whether PANs captured unique variance in neural representations beyond task-optimized models and heuristics, we employed a residualization procedure. For each model type, we regressed out the contribution of alternative models from both the neural and model (upper triangular, flattened) dissimilarity matrices using ordinary least squares:

$$\text{residual} = \text{target} X(X^T X)^{-1} X^T \text{target}$$

where $X^T X$ contains the regressor dissimilarity matrix. We tested two residualization approaches: (1) "one regressor" - randomly sampling a single model instance from the source model class as the regressor, repeated across all model instances to account for sampling variability; and (2) "all regressors" - using all model instances from a given class simultaneously. We report (2) in Fig. 4 as it is a much more severe constraint on the correlation value between networks, so the effect size of information surviving this procedure is as great as possible. This procedure was applied bidirectionally, residualizing PAN models from task-optimized models and vice versa in Fig. 4. In Supplementary Fig. 5, because the Oracle covariate has no variability (as it is based on the ground truth), we do the "one regressor" method to carry over the variability across PAN instances for statistical quantification. Statistical significance was determined through direct parametric pairwise comparisons between residualized correlation values. The asymmetric pattern of residualization results—where PANs maintained high correlation with neural data after removing task-optimized and oracle-covariate models' variance, but not vice versa—indicates that PANs capture fundamental aspects of neural computation not present in the alternative models.

Computational Implementation

All offline analyses were performed in MATLAB. Where parallelization was beneficial, data splits and model simulations were distributed across a high-performance cluster, with each trial or cross-validation fold assigned to a separate worker. For consistency, the same 79 task conditions were used in both neural and RNN analyses, and all time-series were binned at 50 ms. Optimal hyperparameters in machine-learning style network regression or decoding models (e.g., ridge penalty terms) were chosen via nested cross-validation. The final outputs (hidden states, regression weights, decoded trajectories) were stored and evaluated with identical metrics for both neural and model data.

Code and Data Availability

All methods for data preprocessing, model simulations, and decoding analyses were implemented in MATLAB and will be made available through a public repository.

Acknowledgements

We are grateful to Mehrdad Jazayeri for discussions about this project and generously sharing neural data. We also thank Damon Clark and Steve Chang for their comments on this work. We are thankful to the Cognitive and Neural Computational Lab at Yale for their feedback throughout this project, and to Yale Center for Research and Computing for maintaining the HPCs utilized by this project (Misha, Milgram).

References

1. C. R. Gallistel and Adam Philip King. *Memory and the Computational Brain.* 2009.
2. Peter W. Battaglia, Jessica B. Hamrick, and Joshua B. Tenenbaum. Simulation as an engine of physical scene understanding. *Proceedings of the National Academy of Sciences of the United States of America*, 110(45):18327–18332, 11 2013.

3. Máté Lengyel. Marr's three levels of analysis are useful as a framework for neuroscience. *The Journal of Physiology*, 602(9):1911–1914, 5 2024.

4. David Marr. (1982) David Marr, Vision, San Francisco: W. H. Freeman, pp. 19-38, 54-61. In *Neurocomputing, Volume 1.* 2022.

5. Ilker Yildirim, Mario Belledonne, Winrich Freiwald, and Josh Tenenbaum. Efficient inverse graphics in biological face processing. Technical report, 2020.

6. Manthan Khona and Ila R Fiete. Attractor and integrator networks in the brain. *Nature Reviews Neuroscience*, 23:744–766, 2022.

7. Rishi Rajalingham, Hansem Sohn, and Mehrdad Jazayeri. Dynamic tracking of objects in the macaque dorsomedial frontal cortex. *Nature Communications*, 16:346, 2025.

8. Marcelo G. Mattar and Máté Lengyel. Planning in the brain. *Neuron*, 110(6):914–934, 3 2022.

9. François Osiurak and Arnaud Badets. Tool use and affordance: Manipulation-based versus reasoning-based approaches. *Psychological Review*, 123(5):534–568, 10 2016.

10. Rishi Rajalingham, A Piccato, and Mehrdad Jazayeri. Recurrent neural networks with explicit representation of dynamic latent variables can mimic behavioral patterns in a physical inference task. *Nature Communications*, 13:1–15, 2022.

11. Rüdiger Wehner. 'matched filters'—neural models of the external world. *Journal of comparative physiology A*, 161(4):511–531, 1987.

12. Donald D Hoffman. The interface theory of perception. *Stevens' handbook of experimental psychology and cognitive neuroscience*, 2:1–24, 2018.

13. Adrien Doerig, Rowan P Sommers, Katja Seeliger, Blake Richards, Jenann Ismael, Grace W Lindsay, Konrad P Kording, Talia Konkle, Marcel AJ Van Gerven, Nikolaus Kriegeskorte, et al. The neuroconnectionist research programme. *Nature Reviews Neuroscience*, 24(7):431–450, 2023.

14. Jonathan A Michaels, Stefan Schaffelhofer, Andres Agudelo-Toro, and Hansjörg Scherberger. A goal-driven modular neural network predicts parietofrontal neural dynamics during grasping. *Proceedings of the national academy of sciences*, 117(50):32124–32135, 2020.

15. Alexander JE Kell and Josh H McDermott. Deep neural network models of sensory systems: windows onto the role of task constraints. *Current opinion in neurobiology*, 55:121–132, 2019.

16. Daniel LK Yamins and James J DiCarlo. Using goal-driven deep learning models to understand sensory cortex. *Nature neuroscience*, 19(3):356–365, 2016.

17. Robert Geirhos, Jörn Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A. Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence 2020 2:11*, 2(11):665–673, 11 2020.

18. Jason Z. Kim and Dani S. Bassett. A neural machine code and programming framework for the reservoir computer. *Nature Machine Intelligence*, 5(6), 2023.

19. Kelsey R Allen, Kevin A Smith, and Joshua B Tenenbaum. Rapid trial-and-error learning with simulation supports flexible tool use and physical reasoning. *Proceedings of the National Academy of Sciences*, 117(47):29302–29310, 2020.

20. Tomer D Ullman, Andreas Stuhlmüller, Noah D Goodman, and Joshua B Tenenbaum. Learning physical parameters from dynamic scenes. *Cognitive psychology*, 104:57–82, 2018.

21. Pedro A Tsividis, Joao Loula, Jake Burga, Nathan Foss, Andres Campero, Thomas Pouncy, Samuel J Gershman, and Joshua B Tenenbaum. Human-level reinforcement learning through theory-based modeling, exploration, and planning. *arXiv preprint arXiv:2107.12544*, 2021.

22. 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.

23. Kimberly W Wong, Wenyan Bi, Amir A Soltani, Ilker Yildirim, and Brian J Scholl. Seeing soft materials draped over objects: A case study of intuitive physics in perception, attention, and memory. *Psychological Science*, 34(1):111–119, 2023.

24. Ilker Yildirim, Max H Siegel, Amir A Soltani, Shraman Ray Chaudhuri, and Joshua B Tenenbaum. Perception of 3d shape integrates intuitive physics and analysis-by-synthesis. *Nature Human Behaviour*, 8(2):320–335, 2024.

25. WY Bi, AD Shah, KW Wong, BJ Scholl, and I Yildirim. Computational models reveal that intuitive physics underlies visual processing of soft objects. *Nature Communications*, in press.

26. RT Pramod, Elizabeth Mieczkowski, Cyn X Fang, Joshua B Tenenbaum, and Nancy Kanwisher. Decoding predicted future states from the brain's "physics engine". *Science Advances*, 11(22):eadr7429, 2025.

27. Sarah Schwettmann, Joshua B Tenenbaum, and Nancy Kanwisher. Invariant representations of mass in the human brain. *Elife*, 8:e46619, 2019.

28. RT Pramod, Michael A Cohen, Joshua B Tenenbaum, and Nancy Kanwisher. Invariant representation of physical stability in the human brain. *Elife*, 11:e71736, 2022.

29. Aarit Ahuja, Nadira Yusif Rodriguez, Alekh Karkada Ashok, Thomas Serre, Theresa M Desrochers, and David L Sheinberg. Monkeys engage in visual simulation to solve complex problems. *Current Biology*, 34(24):5635–5645, 2024.

30. Shreya Saxena and John P Cunningham. Towards the neural population doctrine. *Current Opinion in Neurobiology*, 55:103–111, 2019.

31. Naama Brenner, William Bialek, and Rob de Ruyter van Steveninck. Adaptive rescaling maximizes information transmission. *Neuron*, 26(3):695–702, 2000.

32. Rishidev Chaudhuri, Berk Gerçek, Biraj Pandey, Adrien Peyrache, and Ila Fiete. The intrinsic attractor manifold and population dynamics of a canonical cognitive circuit across waking and sleep. *Nature Neuroscience*, 22:1512–1520, 2019.

33. Sung Soo Kim, Hervé Rouault, Shaul Druckmann, and Vivek Jayaraman. Ring attractor dynamics in the Drosophila central brain. *Science*, 356(6340):849–853, 2017.

34. David Sussillo and Larry F Abbott. Generating coherent patterns of activity from chaotic neural networks. *Neuron*, 63(4):544–557, 2009.

35. Herbert Jaeger. The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German national research center for information technology gmd technical report*, 148(34):13, 2001.

36. Joshua I. Glaser, Ari S. Benjamin, Raeed H. Chowdhury, Matthew G. Perich, Lee E. Miller, and Konrad P. Kording. Machine Learning for Neural Decoding. *eNeuro*, 7(4):0506–19, 7 2020.

37. Pierre Pouget, Erik E. Emeric, Veit Stuphorn, Kate Reis, and Jeffrey D. Schall. Chronometry of visual responses in frontal eye field, supplementary eye field, and anterior cingulate cortex. *Journal of Neurophysiology*, 94(3):2086–2092, 9 2005.

38. Steven P. Wise, Driss Boussaoud, Paul B. Johnson, and Roberto Caminiti. Premotor and parietal cortex: Corticocortical connectivity and combinatorial computations. *Annual Review of Neuroscience*, 20:25–42, 1997.

39. Seng Bum Michael Yoo, Jiaxin Cindy Tu, Steven T Piantadosi, and Benjamin Yost Hayden. The neural basis of predictive pursuit. *Nature neuroscience*, 23(2):252–259, 2020.

40. Shigeru Obayashi, Tetsuya Suhara, Koichi Kawabe, Takashi Okauchi, Jun Maeda, Yoshihide Akine, Hirotaka Onoe, and Atsushi Iriki. Functional Brain Mapping of Monkey Tool Use. *NeuroImage*, 14(4):853–861, 10 2001.

41. Hamed Nili, Cai Wingfield, Alexander Walther, Li Su, William Marslen-Wilson, and Nikolaus Kriegeskorte. A Toolbox for Representational Similarity Analysis. *PLOS Computational Biology*, 10(4):e1003553, 2014.

42. Aran Nayebi, Rishi Rajalingham, Mehrdad Jazayeri, and Guangyu Robert Yang. Neural Foundations of Mental Simulation: Future Prediction of Latent Representations on Dynamic Scenes. *Advances in Neural Information Processing Systems*, 36:70548–70561, 12 2023.

43. Joshua B Tenenbaum, Charles Kemp, Thomas L Griffiths, and Noah D Goodman. How to grow a mind: Statistics, structure, and abstraction. *science*, 331(6022):1279–1285, 2011.

44. John P. Cunningham and Byron M. Yu. Dimensionality reduction for large-scale neural recordings. *Nature Neuroscience*, 17(11):1500–1509, 10 2014.

45. Jessica B Hamrick, Kevin A Smith, Thomas L Griffiths, and Edward Vul. Think again? the amount of mental simulation tracks uncertainty in the outcome. In *Proceedings of the annual meeting of the cognitive science society*, volume 37, 2015.

46. Christopher Langdon, Mikhail Genkin, and Tatiana A. Engel. A unifying perspective on neural manifolds and circuits for cognition. *Nature Reviews Neuroscience*, 24(6):363–377, 6 2023.

47. Francesca Mastrogiuseppe and Srdjan Ostojic. Linking connectivity, dynamics, and computations in low-rank recurrent neural networks. *Neuron*, 99(3):609–623, 2018.

48. David L. Barack and John W. Krakauer. Two views on the cognitive brain. *Nature Reviews Neuroscience 2021 22:6*, 22(6):359–371, 4 2021.

49. Sam Musallam, BD Corneil, Bradley Greger, Hans Scherberger, and Richard A Andersen. Cognitive control signals for neural prosthetics. *Science*, 305(5681):258–262, 2004.

50. Mehrdad Jazayeri and Arash Afraz. Navigating the neural space in search of the neural code. *Neuron*, 93(5):1003–1014, 2017.

51. Minyoung Huh, Brian Cheung, Tongzhou Wang, and Phillip Isola. The Platonic Representation Hypothesis, July 2024. arXiv:2405.07987 [cs].

52. Ilker Yildirim and L. A. Paul. From task structures to world models: what do LLMs know? *Trends in Cognitive Sciences*, 28(5):404–415, May 2024.

53. Tiago Branco and Peter Redgrave. The neural basis of escape behavior in vertebrates. *Annual review of neuroscience*, 43(1):417–439, 2020.

54. Federico Claudi, Dario Campagner, and Tiago Branco. Innate heuristics and fast learning support escape route selection in mice. *Current Biology*, 32(13):2980–2987, 2022.

55. Elizabeth S Spelke. Précis of what babies know. *Behavioral and Brain Sciences*, 47:e120, 2024.

56. Steven H. Strogatz. NONLINEAR DYNAMICS AND CHAOS: With Applications to Physics, Biology, Chemistry, and Engineering. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*, pages 1–513, 1 2018.

57. Ken Caluwaerts, Francis Wyffels, Sander Dieleman, and Benjamin Schrauwen. The spectral radius remains a valid indicator of the Echo state property for large reservoirs. *Proceedings of the International Joint Conference on Neural Networks*, 2013.

58. Rishi Rajalingham, Kohitij Kar, Sachi Sanghavi, Stanislas Dehaene, and James J DiCarlo. Dynamic mental representations in the human dorsomedial prefrontal cortex. *Nature Neuroscience*, 25(6):758–767, 2022.

Supplementary Material

Details of Programming RNNs

In brief, the framework for programming the weights and connectivity of an RNN [18] is as follows:

1. Define the Network Update Equation

Here, we define our to-be-programmed Reservoir Network (a specific type of RNN) via the update equation, describing how each latent unit evolves in time, as

$$\frac{1}{\gamma}\dot{\vec{r}}(t) = -\vec{r}(t) + \tanh(A\vec{r}(t) + B\vec{x}(t) + \vec{d}) \tag{1}$$

where, given a network with N neurons, M inputs, and P outputs

1. $\gamma \in \mathbb{R}^1$ is the continuous time/rate constant
2. $\vec{r}(t) \in \mathbb{R}^{N\times 1}$ (state vector) describes the value of each neuron within the network at time t
3. $\vec{x}(t) \in \mathbb{R}^{M\times 1}$ are the inputs to the network at time t
4. $A \in \mathbb{R}^{N\times N}$, is the adjacency matrix describing the recurrent neural connectivity. It can be seen as a directed graph that maps neurons to neurons.
5. $B \in \mathbb{R}^{N\times M}$, is the "read-in" matrix describing the exogenous connectivity into the network. It can be seen as a directed graph that maps inputs to neurons.
6. $\vec{d} \in \mathbb{R}^{N\times 1}$, is the constant bias vector for each neuron in the network.

2. Initialize and Solve this Differential Equation

We solve our update equation, which is a differential equation, at a randomly chosen operating point, $r^* \sim \mathcal{U}$, where $\mathcal{U}(-0.5, 0.5)$ is a uniform distribution between $[-0.5, +0.5]$. This yields an approximation of the state vector, $\vec{r}(t)$, as a function of its symbolic inputs and the time derivative(s) of these inputs, yielding

$$\vec{r}(t) \approx h(\vec{x}, \dot{\vec{x}}, \ddot{\vec{x}}, \ldots) \tag{2}$$

3. Decompile into Dynamical Primitives

Once solved at a given operating point, and for a randomly initialized set of connection weights $(A, B) \sim \mathcal{U}(-0.5, 0.5)$ and biases $\vec{d} = \tanh^{-1}(r^*) - Ar^*$, we can use an integral expansion to decompile the network into a set of expansion bases, $R \in \mathbb{R}^{N\times K}$, and a symbolic set of inputs, $\vec{x}_{sym} \in \mathbb{R}^{K\times 1}$, where $h(\vec{x}, \dot{\vec{x}}, \ddot{\vec{x}}, \ldots) \mapsto R\vec{x}_{sym}$. Here we use a multivariate Maclaurin series expansion, $\mathcal{M}_k[h(\vec{x}, \dot{\vec{x}})]$, where k is the order of the expansion, defined as

$$\mathcal{M}_k[h(\vec{v})] = \sum_{m=0}^{k} \frac{1}{m!} \sum_{i_1, i_2, \ldots, i_m = 1}^{2n} \left.\frac{\partial^m h}{\partial v_{i_1} \partial v_{i_2} \cdots \partial v_{i_m}}\right|_{\vec{0}} v_{i_1} v_{i_2} \cdots v_{i_m} \tag{3}$$

where $\vec{v} = [\vec{x}, \dot{\vec{x}}]^T \in \mathbb{R}^{2M}$ concatenates position and velocity terms of our input space:

- $\vec{x} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^T$, $\dot{\vec{x}} = \begin{bmatrix} \dot{x}_1 & \dot{x}_2 & \dot{x}_3 \end{bmatrix}^T$
- $\vec{v} = \begin{bmatrix} x_1 & x_2 & x_3 & \dot{x}_1 & \dot{x}_2 & \dot{x}_3 \end{bmatrix}^T$

This method yields a set of symbolic weights, corresponding to combinatorial/multivariate polynomial coefficients of our symbolic inputs. For example, if we have $M = 3$ inputs to our network, take information up to the first time derivative of our inputs, and want expansion up to order $k = 3$, then:

$$\begin{aligned}\mathcal{M}_3[h(\vec{v})] &= h(\vec{0},\vec{0}) + \sum_{i=1}^{6} \frac{\partial h}{\partial v_i}\bigg|_{\vec{0}}(v_i) + \frac{1}{2!}\sum_{i,j=1}^{6} \frac{\partial^2 h}{\partial v_i \partial v_j}\bigg|_{\vec{0}}(v_i v_j) \\ &\quad + \frac{1}{3!}\sum_{i,j,k=1}^{6} \frac{\partial^3 h}{\partial v_i \partial v_j \partial v_k}\bigg|_{\vec{0}}(v_i v_j v_k) \\ &= \begin{bmatrix} h(\vec{0},\vec{0}) & , & Jh|_{\vec{0}} & , & Hh|_{\vec{0}} & , & T_{i,j,k}h|_{\vec{0}} \end{bmatrix} \\ &\quad \times \begin{bmatrix} 1 \\ (x_1, x_2, x_3, \dot{x}_1, \dot{x}_2, \dot{x}_3) \\ (x_1x_2,\, x_1x_3, x_1\dot{x}_1 \ldots,\, x_2^2,\, x_3^2) \\ \vdots \end{bmatrix} \\ &= R\vec{x}_{sym} \end{aligned} \tag{4}$$

Where $(Jh|_{\vec{0}}, Hh|_{\vec{0}}, T_{i,j,k}h|_{\vec{0}})$ are the Jacobian, Hessian, and third-order terms of our expansion, respectively, and the basis vector $\vec{x}_{sym}$ contains all monomials up to degree 3 in the components of $\vec{v}$; R contains the corresponding coefficients from the Jacobian, Hessian, and third-order derivative tensor evaluated at $\vec{v} = \vec{0}$.

4. Define Recurrence Variables

We next define a set of recurrence variables $\vec{z}(t) \in \mathbb{R}^{P\times 1}$ that encode the desired dynamics of our system. These variables represent the computational state we wish to program into the network. Each recurrence variable z_i is governed by a differential equation of the form:

$$\dot{z}_i = f_i(\vec{z}, t) \tag{5}$$

where f_i defines the dynamics for the i-th variable. For complex dynamical systems, these equations can include:

- **Constant registers**: Variables with $\dot{z}_i = 0$ that maintain initial conditions throughout the simulation
- **Linear dynamics**: Variables following $\dot{z}_i = \alpha z_i + \beta z_j + \ldots$
- **Nonlinear dynamics**: Variables with polynomial or other nonlinear terms, e.g., $\dot{z}_i = \alpha z_i - \beta z_i^3$
- **Coupled dynamics**: Variables whose evolution depends on multiple other variables through complex interaction terms

5. Compile Target Dynamics

Given our target dynamics $\dot{\vec{z}} = f(\vec{z})$, we create a target observation matrix $O \in \mathbb{R}^{P\times K}$ by evaluating the same monomial basis functions used in our expansion on the target dynamics:

$$O = \mathcal{M}_k[f(\vec{z})] \tag{6}$$

This yields a matrix where each row corresponds to a target variable's dynamics expressed in the same basis as our network expansion.

6. Program the Readout Matrix

We solve for the optimal readout matrix $W \in \mathbb{R}^{P\times N}$ that maps the network state to our target dynamics:

$$W^* = \arg\min_W \|WR - O\|_F^2 \tag{7}$$

where $\|\cdot\|_F$ denotes the Frobenius norm. This least-squares problem has the closed-form solution:

$$W^* = OR^\dagger \tag{8}$$

where $R^\dagger$ is the Moore-Penrose pseudoinverse of R.

7. Close the Loop

Finally, we close the loop by partitioning the input matrix B into recurrent and exogenous components:

$$B\vec{x} = \hat{B}\hat{\vec{x}} + \bar{B}\bar{\vec{x}} \tag{9}$$

where $\hat{\vec{x}} = W\vec{r}$ represents the recurrent feedback and $\bar{\vec{x}}$ represents any external inputs. This yields the programmed network:

$$\frac{1}{\gamma}\dot{\vec{r}}(t) = -\vec{r}(t) + \tanh(A^*\vec{r}(t) + \bar{B}\bar{\vec{x}}(t) + \vec{d}) \tag{10}$$

where $A^* = \hat{B}W$ is the effective adjacency matrix encoding our target dynamics.

This framework enables the analytical embedding of arbitrary dynamical systems into recurrent neural networks, transforming differential equations into neural connectivity patterns that autonomously execute the desired computations. The reader can consult ref. [18] for further details.

8. Initialize and Simulate

To simulate a specific instance of a Pong condition:

1. Set initial conditions $\vec{z}_0$ through the input channel and evolve the network from the global operating point until convergence ($r_i^*(t) - r_i^*(t-1) < \epsilon$). This establishes the conditional operating point corresponding to a particular Pong condition:

$$\vec{r}_i^* = \tanh(B\vec{z}_0 + \vec{d}) \tag{11}$$

2. Numerically integrate the **programmed network** equation from $\vec{r}_i^*$ using standard ODE solvers (e.g., Runge-Kutta methods)

$$\frac{1}{\gamma}\dot{r} = -r + \tanh(Ar + Bx + d) \tag{12}$$

   where we begin at $r = r_i^*$ to simulate a Pong condition with the programmed adjacency $A = A^* = \hat{B}W$.

3. Read out the observable state through the programmed readout: $\vec{z}(t) = W\vec{r}(t)$

Pseudo-Code of the Board Simulation

716

a. Object-Oriented Programming Constructor for the Simulation

```
obj BoardState(self, ballX = float, ballY = float, velX = float, velY = float)

Initial conditions (provided by external input):
    self.ballX = ballX;
    self.ballY = ballY;
    self.velX  = velX;
    self.velX  = velX;

Initial state (determined by the initial conditions):
    self.velXSample      = sign(velX)
    self.velXReflection = −sign(velX)
    self.velYSample      = sign(velY)
    self.velYReflection = −sign(velY)
    self.colLeft      = 0
    self.colRight     = 0
    self.colTop       = 0
    self.colBottom  = 0

    return state
```

b. Main Function for Running the Simulation

```
def main( tMax, ballx,bally,velX,velY)
    state = BoardState(ballX = float, ballY = float, velX = float, velY = float)
    t = 0
    while t < tMax
        state = colLeft(state)
        state = colRight(state)
        state = colTop(state)
        state = colBottom(state )

        state = updateX(state )
        state = updateY(state )

        t += 1
    end
```

c. Pseudo-Code Analogy of Dynamical Recurrent Logic

Collision Detection

```
def colRight(state):
        if (state.ballX - wallRight) < err
            flipSignXRight(state)
        return state

def colLeft(state):
        if (state.ballX - wallLeft) < err
            flipSignXLeft(state)
        return state

def colTop(state):
        if (state.ballY - wallTop) < err
            flipSignYTop(state)
        return state

def colBottom(state):
        if (state.ballY - wallBottom) < err
            flipSignYBottom (state)
        return state
```

Non-linear Velocity Flip if Collision is Detected

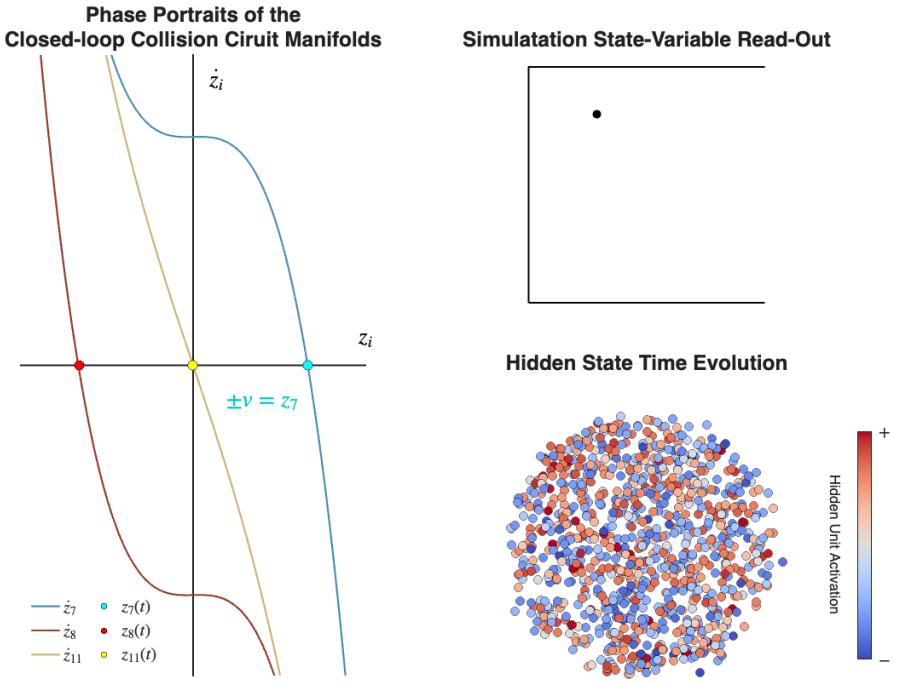
```
def flipSignXRight(state):
        state.velXSample    =  −state.velXSample;
        flipSignXLeft(state)
        return state

def flipSignXLeft(state):
        state.velXReflection = −state.velXReflection
        flipSignXRight(state)
        return state

def flipSignYTop(state):
        state.velYSample = −state.velYSample
        flipSignYBottom(state)
        return state

def flipSignYBottom(state):
        state.velYReflection = −state.velYReflection;
        flipSignYTop(state)
        return state
```

Linear Position Update Using Hysteretic Velocity State

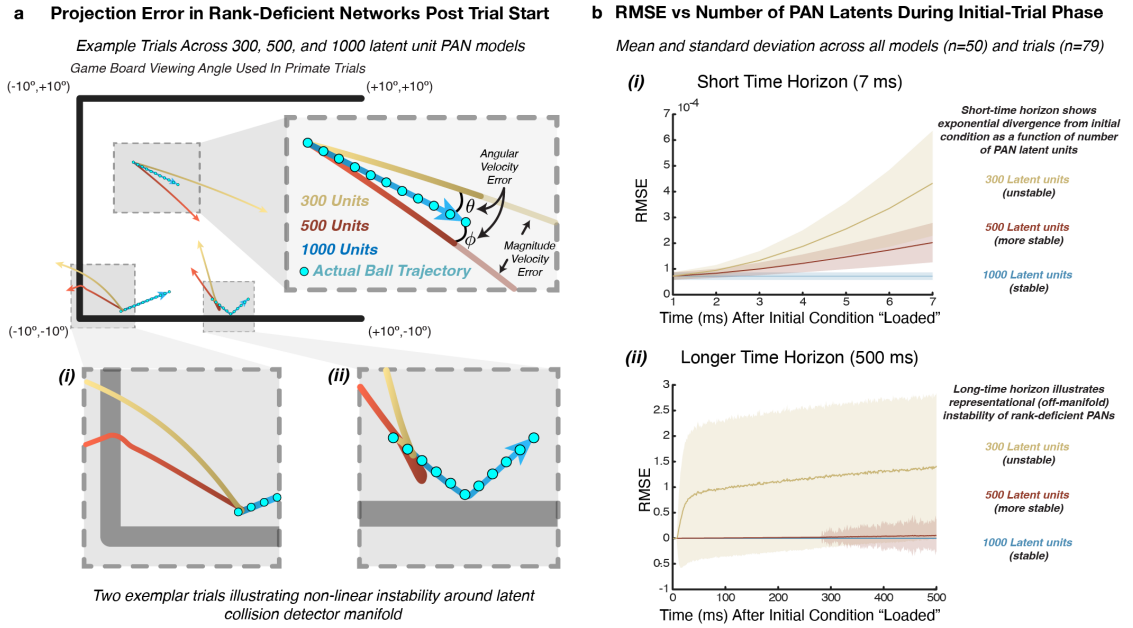
```
def updateX(state):
        state.ballX = state.ballX+ (self.velXSample)state.velX;
        return state

def updateY(state):
        state.ballY = state.ballY + (self.velYSample)state.velY;
        return state
```

Supplementary Figure 1. Pseudocode of the symbolic program underlying the dynamical attractor manifold embedded in the PAN model. This figure presents an intuitive pseudocode representation to illustrate how the 12 coupled differential equations (Equations $(\dot{z}_1)$ through $(\dot{z}_{12})$) described in the Methods function together to simulate the board dynamics. (a) An object-oriented programming constructor of the board state, showing how initial conditions map to the initial (x, y) position of the ball within the game board and the constant velocity registers (z_1,z_2 from equations $\dot{z}_1$,$\dot{z}_2$ in Methods) and initialize the hysteretic state variables. (b) Main loop for unfolding the board dynamics. (c) Pseudocode modules corresponding to the computational blocks 2-4 of the Methods section: Collision Detection (yellow) represents the dynamics of z_9-z_{12} (Equations $(\dot{z}_9)$-$(\dot{z}_{12})$ in Methods), which implement threshold-based collision detection; Non-linear Velocity Flip (blue) illustrates the hysteretic NAND gate dynamics of z_5-z_8 (Equations $(\dot{z}_5)$-$(\dot{z}_8)$ in Methods) that maintain velocity sign memory through bistable attractors; Linear Position Update (purple) shows how z_3 and z_4 (Equations $(\dot{z}_3)$ and $(\dot{z}_4)$ in Mehtods) integrate velocity to update position. These functions make recurrent calls to each other.In PAN, these dynamics are analytically embedded into the recurrent connectivity matrix $A = \hat{B}W$, allowing the network to autonomously execute these computations through its natural dynamics.

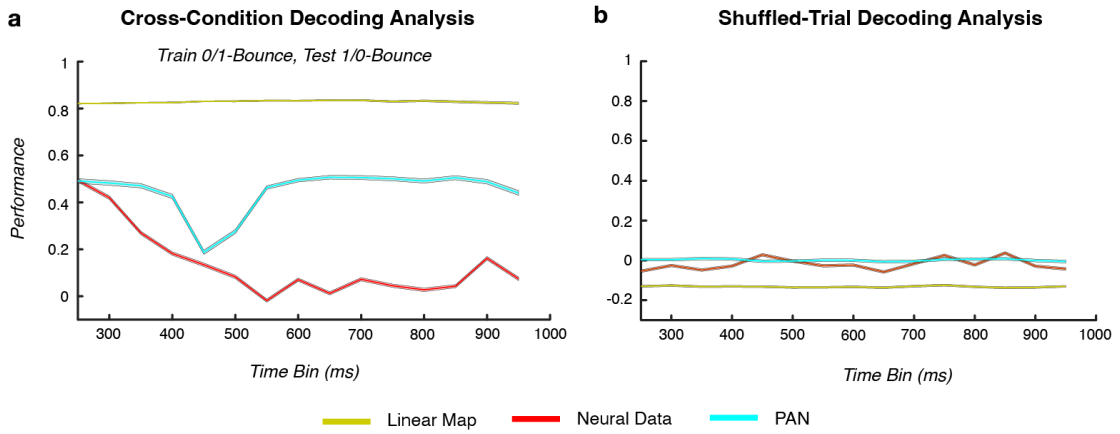
Animation of the Dynamical Attractor Manifold and PAN hidden state



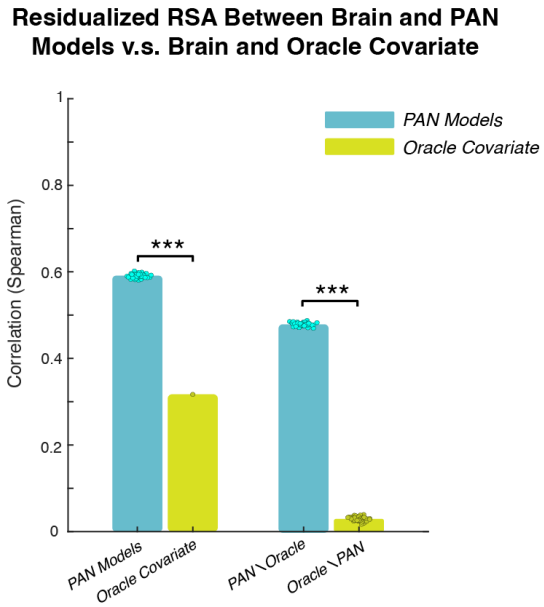
Supplementary Figure 2. (NOTE to reader: To play the animation within the PDF, please use Adobe Acrobat.) **Real-time visualization of the programmed dynamical attractor manifold implementing non-linear velocity reflection in a 1000-neuron PAN during wall collision.** Left panel: Phase portraits of the collision circuit's recurrent dynamics, showing the theoretical nullclines (continuous curves) programmed via the polynomial-shift operators and pitchfork bifurcations described in Methods (equations $\dot{z}_7$-$\dot{z}_8$, and $\dot{z}_9$, which recurrently couple to z_2 the symbolic representation of the y-position of the ball). The points represent the actual instantaneous values of these dynamical variables as read out from the network state via $Wr(t)$. The blue and red curves show the hysteretic switch's pair-dynamics ($\dot{z}_7$ and $\dot{z}_8$ for vertical velocity), implementing bistable memory. The yellow curve represents the collision detector, which transitions between stable zero and non-zero fixed points when the ball approaches the boundary. The cross-coupling between collision detection and velocity sign variables implements the NAND gate logic triggering velocity reversal upon collision. Top right panel: Real-time ball trajectory on the board, computed from the linear position update equations ($\dot{z}_3, \dot{z}_4$) that integrate the velocity components stored in the hysteretic switches. The red trace shows the ball's path. The wall collision triggers a transition in the dynamical landscape (left panel), flipping the appropriate velocity component while maintaining the orthogonal component unchanged. Bottom right panel: High-dimensional hidden state of PAN showing all $N = 1000$ neuron activations $r(t) \in \mathbb{R}^N$ at each time point. The color gradient (from dark blue to light cyan) encodes individual neuron activities. Despite this high-dimensional embedding, the network autonomously constrains its dynamics to the 12-dimensional manifold specified by the programmed differential equations. The L_2-norm solution that creates the programmed adjacency ($A = \hat{B}W$) during the compilation step, distributes the low-rank symbolic/dynamical representations heterogeneously across the population-rank of the network. The relatively uniform distribution of activities indicates that the network utilizes its full representational capacity rather than sparse coding, consistent with such a L_2-norm based analytical encoding.



Supplementary Figure 3. Stability analysis of PANs with varying numbers of hidden units. (a) Projection error visualization in networks after trial start, showing example trajectories from PANs with 300 (yellow), 500 (red), and 1000 (blue) hidden units. The main panel displays ball trajectories overlaid on the board, with cyan dots indicating the actual ball trajectory. Two types of errors emerge: angular velocity errors (θ,ϕ) and velocity magnitude errors (trajectories going farther/faster than ground truth in the same amount of time). Insets (i) and (ii) show zoomed views of critical trajectory segments where non-linear instabilities manifest around the latent collision detector manifold, visible in the under-parameterized 300 and 500 unit networks' trajectories. (b) Root Mean Square Error (RMSE) quantification as a function of PAN size during the initial trial phase, averaged across 50 models and 79 conditions per model size. (i) Short time horizon (7 ms) reveals exponential divergence from initial conditions, with error growth inversely proportional to network size. The 300-unit network (unstable) shows rapid divergence, while the 1000-unit network remains stable. (ii) Longer time horizon (500 ms) demonstrates the representational consequences of rank deficiency, where off-manifold instabilities in smaller networks lead to recurrent propagation of trajectory errors. The 1000-unit network maintains near-zero RMSE throughout, indicating sufficient expressivity to stably embed the 12-dimensional dynamical manifold, while the 300 and 500-unit networks exhibit persistent drift from the programmed dynamics.



Supplementary Figure 4. Heuristic strategies versus dynamical computation in DMFC. **a**, To distinguish between the Linear Map heuristic and PAN's nonlinear dynamical attractors, we leveraged a critical difference in their predictions. A linear mapping from the ball's current position and velocity to its final position represents a simple heuristic solution to the interception task — one that does not require simulation of intermediate dynamics and that is accurate due to the simplicity of the board (linear or piecewise linear trajectories across 0-bounce and 1-bounce conditions). Decoders trained on 0-bounce conditions and tested on 1-bounce conditions (and vice versa) reveal fundamentally different computational strategies between the Linear Map versus PAN and DMFC. All of these decoders are trained on a given time point (along the x-axis) to predict the ball end-point (as in Fig. 2b analysis). The Linear Map heuristic from ground truth $(x, y, \Delta x, \Delta y)$ (yellow) maintains high decoding generalization (~ 0.8 correlation) throughout the trial.In contrast, both neural data (red) and PANs (cyan) show poor initial generalization that worsens over time, with correlation dropping below 0.2 within 500 ms. We also note an important difference between PAN and DMFC: In PAN, the decoder's generalization performance improves quickly past 500 ms, whereas in DMFC it either stays flat or improves very gradually. Together, these results demonstrate that despite the availability of a simpler heuristic, DMFC implements trajectory-specific nonlinear dynamics. **b**, A separate, heuristic possibility is that the end-point decoding is due to some spurious correlations in the dataset. To rule out this possibility, we train and test ball end-point decoders with the condition labels randomly shuffled. When trial labels are randomly shuffled, all three decoders fail (correlations near zero), confirming that successful decoding is not due to spurious correlations throughout our analysis. Error bands represent SEM.



Supplementary Figure 5. Correlation (Spearman's ρ) between the representational similarity matrices of neural data and PAN models vs neural data and the Oracle covariate (i.e., ground-truth position and velocity of the ball). PAN models (cyan) achieve significantly higher correlations with neural data compared to the Oracle covariate (yellow) (PAN: $\rho = .59$ versus Oracle: $\rho = .32; p < .001$). After residualizing PAN's representational similarity values from the Oracle covariate, this covariate nearly decouples from the neural data ($\rho = 0.03, p = .002$). In contrast, after residualizing the Oracle covariate, PAN maintains a high correlation with neural data ($\rho = 0.48, p < .001$). This asymmetry indicates that structure-preserving RNNs capture fundamental aspects of neural computation not present in just the position and velocity of the ball, suggesting PAN models accurately capture representational structure shared with the brain. (***: $p < .001$)