Hypothesis-Driven Identification of Neural Algorithms With Dynamical Structure-Preserving Manifolds

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

Neuroscience aims to uncover the algorithms by which the brain builds and manipulates complex internal representations of objects, agents, and places. Existing reverse-engineering frameworks remain ineffective for identifying these algorithms. Here, we present Dynamical Structure-Preserving Manifolds (dSPMs) — a new reverse-engineering framework that unifies symbolic, structure-preserving abstractions with dynamical systems to enable hypothesis-driven identification of neural algorithms undergirding mental representations. We use dSPM to identify a physical prediction algorithm in the dorsomedial frontal cortex (DMFC) of macaques intercepting a ball in a Pong-like gameboard. dSPM posits that the perceived initial conditions of a scene quickly collapse neural activity onto a low-dimensional manifold, whose geometry corresponds to a structure-preserving physics-based representation of the gameboard. dSPM better explains DMFC than alternatives, makes predictions confirmed in neural data, and suggests prediction occurs not through next-step simulation or task-performant heuristics, but via the physics-based topological structure of this manifold. dSPM offers a tool for effectively exploring the computational foundations of biological intelligence.

Introduction

The brain generates rich internal representations of the world around us, enabling predictions about how objects will move, agents will act, and places will connect. A central challenge of neuroscience is to uncover the algorithms — i.e., the precise, if possible interpretable, computational transformations — by which the brain builds and manipulates these complex mental representations.

Two reverse-engineering approaches dominate current work, each focusing on different perspectives of cognition and featuring different advantages and limitations. On one hand, probabilistic models of cognition specify interpretable hypotheses about mental representations [1], but typically lack neural grounding. These models suggest that the brain involves structure-preserving representations of the external data-generating processes [2] — e.g., the physics that govern how objects move and react to external forces —that in turn support predictions of what will happen next [3–5]. However, these hypotheses are typically formulated as "computational-level" or functional accounts of the brain [6, 7]: i.e., in practice, they are implemented using standard computer software and high-level programming languages, without identifying an actual algorithm that may realize them in biological neural circuits. On the other hand, deep neural network (DNN) models fit neural

data with remarkable precision but obscure the algorithms they learn. DNNs facilitate explorations of task-optimized statistical representations as accounts of mental representations [8–10], which can involve "shortcuts" and "simple tricks" that are typically, but not always, performant [10, 11]. Crucially, DNNs offer scientific hypotheses with respect to their architectures, training data, and training objectives [12], and not directly with respect to the possible algorithms themselves. Instead, these "implicit algorithms" need to be "extracted" after the fact using various interpretation tools, regardless of whether these DNNs are large or small (e.g., [13, 14]).

This landscape of modeling approaches highlights a critical gap: intuitive cognitive theories that are difficult to map to neurobiology, and powerful fits to neural data that do not allow algorithmic-level control over hypotheses. A new approach is needed to precisely and interpretably identify neural algorithms of complex mental representations.

Here, we introduce Dynamical Structure-Preserving Manifolds (dSPM) for hypothesis-driven identification of neural algorithms and present a case study of physical scene prediction in macaque frontal circuitry to establish its efficacy. At its core, dSPM unifies two influential but, to this day, disparate views of the brain: that the brain can be viewed as a symbolic representation system building and manipulating structure-preserving mappings of the external data-generating processes [2, 15] (Fig. 1A) and that the brain can be viewed as the evolution of coupled dynamical systems in population state space [16, 17] (Fig. 1B).

The dSPM framework achieves this synthesis in two stages. First, a symbolic representation hypothesis (e.g., a program-like specification of objects and their interactions) is captured in a dynamical algorithm. This dynamical algorithm is a system of coupled ordinary differential equations with appropriately expressive computational primitives to fully translate variables and functions in the symbolic representation. Even though dynamical systems offer a common framework for neuroscience, this framework is most often used to interpret or analyze otherwise high-dimensional activity (e.g., [18, 19]). The dSPM framework's "top-down" construction of computation through dynamics, by specifying a dynamical algorithm of complex representations, contrasts with this traditional approach.

Second, dSPM embeds this dynamical algorithm within a reservoir computer, a biologically plausible architecture for recurrent, distributed computation [20], by analytically determining the connectivity (i.e., the adjacency matrix) of the reservoir computer without any training, training data, or training objectives. The result is a population of interconnected units whose analytically-set connectivity directs the flow of information to align with the dynamical algorithm of the hypothesized symbolic representation. Accordingly, dSPM fills the modeling gap mentioned earlier: Unlike task-optimized DNNs, dSPM's reservoir computer is a white-box at the algorithm level, and unlike probabilistic models of cognition, dSPM is falsifiable by neural data, enabling hypothesis-driven exploration of the neural algorithms of complex mental representations.

To assess the efficacy of dSPM, we apply it to identify an algorithm for physical scene prediction in macaque frontal circuitry. Rajalingham et al. [21] performed high-throughput single-cell recordings in the dorsomedial frontal cortex (DMFC) of macaques performing a virtual ball interception task in a Pong-like gameboard. The researchers occluded the ball's trajectory partway through the trial, finding evidence of prediction in the DMFC activity. How does the DMFC population represent the gameboard, and how does this representation support the prediction of future trajectory? The dSPM framework enables a new reverse-engineering capability: It allows us to test the possibility of a physics-based, structure-preserving representation of the gameboard — including positions of the ball and the walls, velocity, and collision state, as well as interaction rules according to Newtonian mechanics (as in the cognitive modeling of ref. [3]) — but directly in the high-dimensional neural activity. We implement dSPM for this domain using a recent methodology from the physics of dynamical systems [22], which provides a dynamical algorithm of a physics-based representation of the gameboard as well as a general-purpose analytical method for translating this dynamical algorithm into the connectivity of a reservoir computer. We evaluate dSPM's reservoir computer against the DMFC activity, while making comparisons to powerful alternative models, including a series of task-optimized recurrent neural networks (whose task performance is previously shown to

correlate with macaque behavioral outputs in the same task [23]) and a task-performant heuristic specialized to this ball-interception task.

The dSPM model makes a surprising prediction about the DMFC activity, which we call "single-state sufficiency for entire trajectory prediction": The perceived initial conditions of a scene should rapidly bias the high-dimensional neural state toward a low-dimensional manifold whose physics-based topological structure deterministically encodes future trajectory information, without a need for step-by-step simulation. Remarkably, we confirm this prediction in the DMFC activity: within ~ 250 ms, approximately by the time visual information reaches the frontal cortex [24], the entire future trajectory of the ball becomes linearly decodable in the DMFC population activity. Moreover, using representational similarity analysis [25], we show that dSPM explains statistically significant and substantial variance in the neural data, subsuming nearly all of the variance that can be explained by alternative models. These results support the hypothesis that DMFC implements a structure-preserving physics-based representation of physical scenes and enables prediction via rapid manifold collapse. This demonstrates dSPM's efficacy in hypothesis-driven exploration of the neural algorithms of complex mental representations.

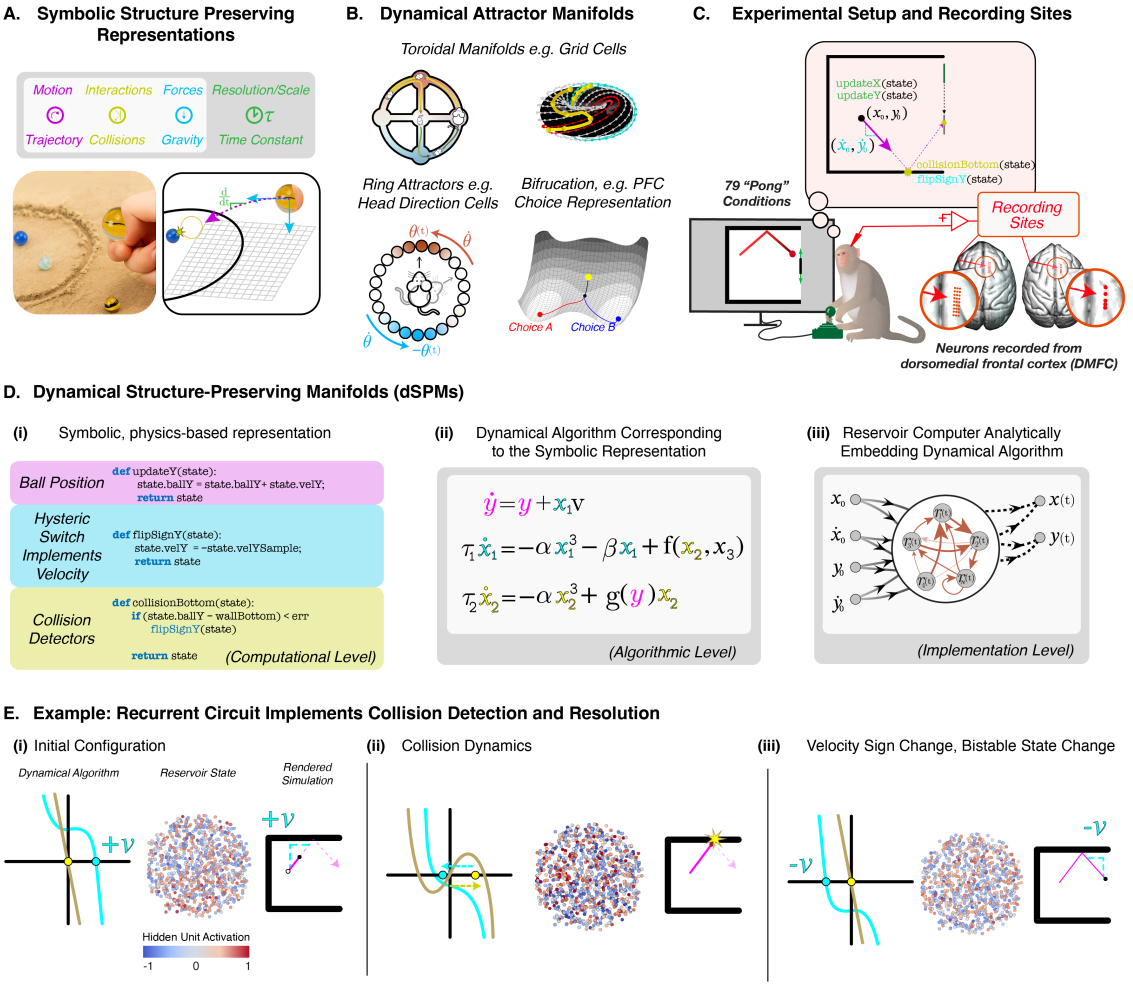


Figure 1. Overview of the Dynamical Structure-Preserving Manifolds framework. (A) The brain can be viewed as a symbolic representation system with structure-preserving mappings of the external data-generating processes, including the physics that govern how objects move and interact. (B) Example dynamical systems in neuroscience implementing complex cognitive representations, including a ring attractor for heading direction, a torus for position in 2D space, and bifurcation for decision-making. (C) Experimental design showing neural recordings from DMFC in two macaques performing a ball interception task in a Pong-like gameboard. The "thought bubble" indicates the hypothesis dSPM enables us to test: DMFC circuitry builds and runs forward a physics-based representation of the gameboard, including the entities and relations of the ball, walls, and the paddle. (D-i) The dSPM framework starts a computational-level, program-like description that functionally captures the physics-based representation of the gameboard to support trajectory prediction, but without neural grounding. (D-ii) Its dynamical algorithm, i.e., a set of coupled differential equations, corresponding to the program-like representation in panel (D-i). Further explained on panel (E). (D-iii) The dSPM's reservoir computer analytically embeds the dynamical algorithm in its connectivity matrix, without training, training objective, or training dataset. This reservoir computer is a neurally falsifiable instantiation of the physics-based representation hypothesis. (E) Collision detection and resolution in dSPM through dynamics and recurrent circuits that undergo phase transitions when the ball approaches a wall. In each subpanel, leftmost line plots are the phase portrait of the two coupled (out of 12) dynamical variables (cyan: velocity; yellow: collision detection); the middle scatterplots show the hidden units of the reservoir computer; the rightmost panels visualize the gameboard. (E-i) Unfolding the initial configuration by integrating a positive velocity value; (E-ii) Collision detection and resolution via cubic bifurcation (yellow line), which non-linearly bifurcates from one stable attractor in (E-i) to two stable attractors (outer zero-crossings) and one repeller (middle zero-crossing) due to proximity to the wall. This transition pulls the velocity variable, causing a sign flip (blue line). (E-iii) Unfolding the rest of the simulation, similar to (E-i), but by integrating this negative velocity value $-v$.

Results

Ball interception task

To apply dSPM to identify a neural algorithm of physical scene prediction, we leveraged a virtual ball-interception task, developed by Rajalingham et al. [26, 27]. In this task, rhesus macaques (*Macaca mulatta*) use a joystick to control a paddle to intercept a moving ball on a computer screen (Fig. 1C), 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 population activity contained information about the ball position regardless of whether it was occluded, making it a prime candidate for implementing physical scene prediction [27].

Dynamical Structure-Preserving Manifolds

How does the DMFC circuitry represent the gameboard and support trajectory prediction? The dSPM framework enables a previously unavailable modeling capability for answering this question: neurally falsifiable models that implement symbolic structure-preserving representations of the physical world. Cognitive theories explore the idea that physical prediction corresponds to building

and running forward a structure-preserving representation (i.e., a homeomorphic map; see [28]) of the worldly causes that shape the way scenes unfold — i.e., an internal representation of approximate Newtonian mechanics [29] (Fig. 1C "thought bubble"). Functionally, structure-preserving representations are appealing: Having access to the worldly causes underlying a scene is behaviorally efficacious [28], supporting efficient learning, flexible generalization [30–32], and compositionality [33]. But their typical implementations [3–5, 34], which involve off-the-shelf computer graphics software and high-level programming languages, do not inform how biological neural systems could implement physics-based, structure-preserving representations of objects.

Following this cognitive modeling work [3], we can express a physics-based representation of the gameboard at an abstract computational-level by the properties and relations of the entities on the board (ball, paddle, walls), including the ball's position and how it changes over time (linear dynamics), if-else branching for collision detection and force relations (Fig. 1D-i; see Methods and Supplementary Fig. 1 for a full description). What neural mechanisms could ground such structure-preserving physics-based representations of scenes in the brain?

The dSPM framework builds on an important clue from the recent work in neuroscience: The geometry of low-dimensional, latent manifolds underlying neural populations is found to correspond to basic structure-preserving representations of the world [17] (Fig. 1B), such as ring attractor and toroidal geometry for heading direction and position in 2D space, respectively [17, 35–38]. 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 [37, 38]. It remains unclear how to construct manifolds that compute complex mental representations — beyond the relatively simple domains that have so far been explored.

This brings us to the algorithmic-level formulation of dSPM: We suggest that in the brain, these physics-based representations are encoded in latent dynamical algorithms (Fig. 1D-ii). A dynamical algorithm is formulated by coupling multiple dynamical variables through the coefficients that determine their dynamics. Unlike the common fixed-point attractors [39, 40], this coupling defines a "dynamical manifold" — a manifold whose attractors and repellers not only change over time, but can be computationally harnessed as dynamical variables to build algorithms (cf. [41]). These dynamical manifolds can express symbolic, program-like primitives, including branching-dependent computations such as collision detection and resolution. To create these primitives, we leverage controlled bifurcations as introduced in ref. [42]. For example, by defining the standard cubic bifurcation $\dot{x} = g(t)x - ax^3$, 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 $g(t)$. To algorithmically specify collision detection and resolution, we exploit this property (Supplementary Fig. 2A): 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. 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. The dSPM couples additional differential equations (Fig. 1D-ii), fully specifying a dynamical algorithm in 12 equations that corresponds to the symbolic representation of the gameboard in Fig. 1D-i (see Methods and Supplementary Fig. 2, 3 for the full dynamical algorithm). Much like a ring attractor captures the structure of heading direction, the geometry of the resulting manifold functionally maps a physics-based representation of the gameboard.

To make this dynamical algorithm testable/falsifiable in neural data, dSPM compiles it into a tanh-activated reservoir computer (Fig. 1D-iii) [20, 43]. Following ref. [22], we analytically set the reservoir computer's connectivity weights such that the network's population dynamics align with the dynamical algorithm, without training, training data, or training objectives. In brief, this method "decompiles" the reservoir computer's hidden state and dynamics into an analytical basis of its inputs and uses this basis to program the dynamical algorithm (the set of ordinary differential equations) into the adjacency matrix of the reservoir computer (see Methods and Supplementary

Information). The result is what we call a flow-matched embedding: When we project the reservoir computer's activity, which is much higher dimensional than the dynamical algorithm (1000 units in the reservoir computer vs. 12 dynamical variables in the dynamical algorithm), onto the dynamical variables, the projected dynamics follow the same trajectories prescribed by the dynamical algorithm. In dynamical systems terms, this is a topologically semi-conjugate relationship — the reservoir computer approximates the same flow field as the dynamical algorithm and thus underlying the symbolic representation, though in a higher-dimensional space (Fig. 1E; see Supplementary Fig. 3 for a dynamic visualization).

This completes our description of our dSPM model — a novel reverse-engineering tool that embeds the cognitive hypothesis of a structure-preserving, physics-based representation on one end and is falsifiable in high-resolution neural data on the other. To compare dSPM 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.

Single-state sufficiency mechanism for rapid ball endpoint prediction

The dSPM's reservoir computer, by construction, embeds a lower-dimensional manifold whose geometry corresponds to a physics-based representation of the scene. Once configured with the initial conditions of the gameboard (e.g., initial position and velocity of the ball), this manifold can be projected into the future without any additional sensory input (Fig. 2A). Ordinarily, this can be accomplished via a step-by-step simulation of the reservoir computer, which corresponds to unfolding the hidden state of the reservoir computer using its recurrent connectivity weights. However, future predictions, including the far-out trajectory of the scene, can also be linearly read out of these hidden states. This is because each momentary hidden state contains information not only about the current ball position and velocity but also all 12 dynamical variables that determine the local curvature of the manifold. Thus, if the physics-based representation truly determines the flow field of the dynamical system, the properties of the local vector field at any point should specify global trajectories once the initial conditions provide sufficient constraint [44, 45].

We turn to a striking feature of the DMFC population activity to make an initial test of this single-state sufficiency mechanism. Rajalingham et al. [27] found that DMFC quickly encodes the ball's final position where it is intercepted or exits the gameboard ("ball endpoint") by a mere 250 ms after trial onset. To replicate this result and analyze both the neural data and the models, we created our own decoding pipeline (Fig. 2B). For each trial, we constructed a neural or model state matrix at individual time bins (for a bin size of 50 ms), then used a generalized linear model (GLM) [46] to decode the ball endpoint (see Methods). Fig. 2C (left) shows our replication of Rajalingham et al.'s finding [27] of rapid ball endpoint encoding in DMFC. Rajalingham et al. [27], based on the inability of the models they tested to explain their data, concluded that the brain may be operating on two different "strategies": an offline prediction of task-relevant information (ball endpoint) that occurs early in the trial, with an unspecified mechanism, and an online next-time step prediction in the rest of the trial.

The dSPM model offers the dramatically different possibility of single-state sufficiency: A low-dimensional latent manifold embedded in DMFC renders the ball endpoint a linear projection along the surface of this manifold. Remarkably, applying our neural decoding pipeline to dSPM provides evidence for this possibility, recapitulating the rapid prediction ability of the DMFC with high fidelity (Fig. 2C, middle). The correlation between the decoded ball positions from dSPM'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 (at which point correlation decreases due to the smaller number of Pong conditions long enough for decoding analysis). In an additional, 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 (Supplementary Fig. 5). This finer-grained analysis provides further evidence of the consistency between the DMFC and the dSPM model: Both in the neural data and dSPM, we find robust ball endpoint prediction within

each bounce condition, with slightly better performance for zero-bounce trials.

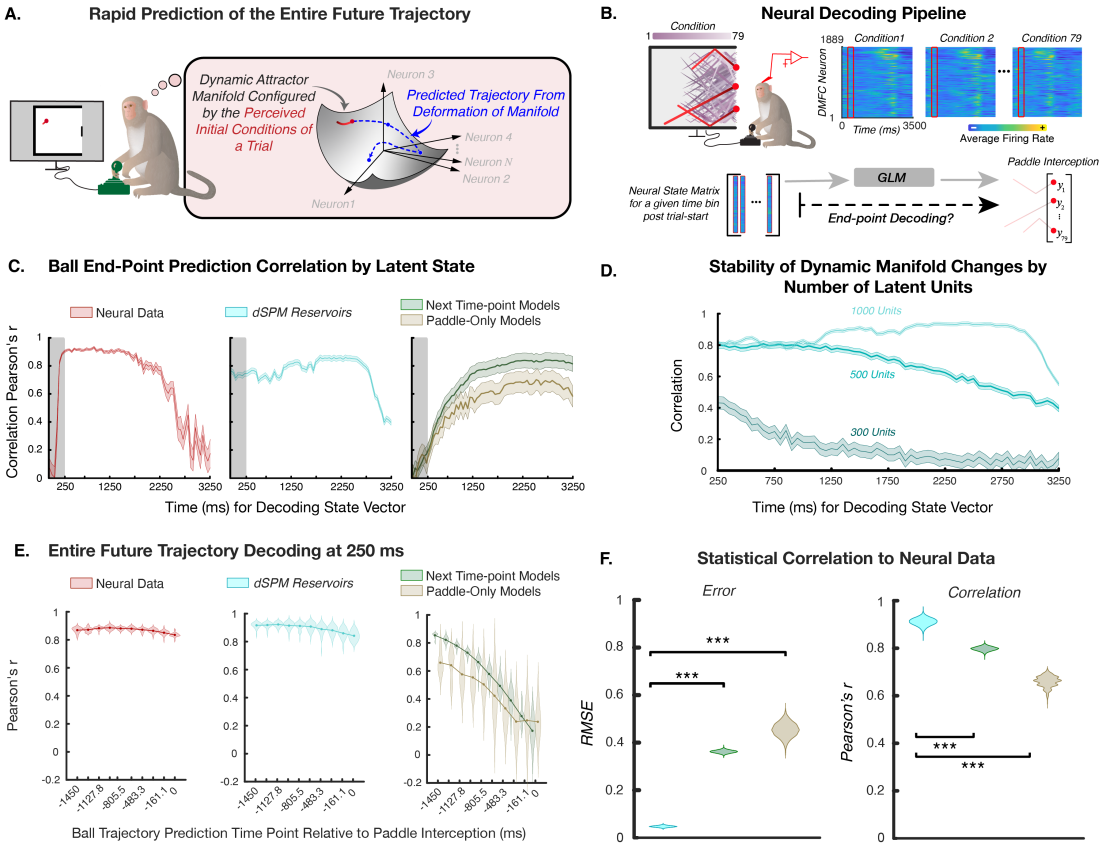


Figure 2. Confirmation of dSPM's single-state sufficiency mechanism of physical prediction in DMFC activity. (A) dSPM posits that the perceived initial configurations of the gameboard bias the DMFC activity toward a low-dimensional manifold whose geometry corresponds to a physics-based representation of the gameboard. This manifold renders points along the future trajectory of the ball lawfully decodable. (B) Our decoding pipeline 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 of 50 ms (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 endpoint"). (C) Temporal evolution of decoding accuracy for the ball endpoint from DMFC, dSPM, and task-optimized RNNs. DMFC neural activity (left, red) shows early prediction capacity ($\sim$ 250ms after trial onset), with high correlation (Pearson's $r > 0.8$) maintained until late in the trial. This pattern is closely matched by dSPM (middle, cyan), while task-optimized RNNs (right, green/brown) show slowly improving prediction accuracy over time. (D) Stability of dynamical attractors is necessary for persistent ball endpoint prediction over time in dSPM. dSPMs with more units (1000 vs. 500 vs. 300) more accurately simulate scenes (Supplementary Fig. 4) and maintain stable predictions of ball endpoint for more extended periods. (E) 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 at 250 ms after trial onset. This confirms the striking prediction of dSPM: at the trial start, the complete future trajectory of the ball, not just the endpoint, is decodable in DMFC (Pearson's $r > 0.8$ throughout time for DMFC and dSPM), but not in task-optimized RNNs. (F) Violin plots and statistical comparisons of the entire future trajectory decoding performance of models. dSPM achieves significantly lower error rates (left) and higher correlation (right), relative to task-optimized RNNs in matching DMFC.

Crucially, in DMFC, this rapid prediction ability of the ball endpoint emerges 250 ms after trial onset, whereas in dSPM (which is provided with the initial configurations of trials), it is immediate (Fig. 2C left vs. middle). This is consistent with the 200-250 ms signal transduction time to this frontal region [24], accounting for the time to "initialize" the DMFC manifold with the initial configuration of the gameboard. This suggests that as soon as the perceived initial configuration of the gameboard is available to DMFC, its neural activity is biased toward an dSPM-like low-dimensional manifold whose geometry corresponds to a physics-based representation of the scene.

An additional prediction of the single-state sufficiency mechanism is that this ability to linearly decode far-out future states should gracefully degrade with the degrading accuracy of the manifold embedded within the reservoir computer. Testing this prediction in the DMFC activity would require perturbation techniques [47] to modulate the accuracy of the neural manifold, which future research should explore. But we can readily test this possibility in the dSPM model. Specifically, we vary the accuracy of the manifold by reducing the number of hidden units and thus the computational capacity of the reservoir computer. Bigger reservoir computers more accurately simulate the gameboard (Supplementary Fig. 4), suggesting that sufficient dimensionality is required to approximate the physical scene. Crucially, we found that these larger networks also maintained stable predictions of ball endpoint for longer periods (Fig. 2D), similar to the DMFC population. This result provides insight into the computational requirements — the accuracy of the manifold in encoding physics-based representations — for robust physical scene prediction in biological neural networks.

Confirming the entire future trajectory prediction of dSPM

These results, focusing on the rapid prediction of the task-relevant ball endpoint information, lead to a remarkable prediction about the DMFC activity: Because the ball endpoint is not coded in any special way in the structure-preserving representation of the gameboard within dSPM, we predict that at the start of the trial, not only the endpoint but also the *entire future trajectory* of the ball will be linearly decodeable in the neural data. In other words, a manifold configured by the perceived initial conditions of the gameboard should make not just the ball endpoint linearly decodable, but also the rest of the trajectory in between. Remarkably, we confirm this prediction in both the DMFC activity and dSPM. In the DMFC, when decoding from neural state at 250 milliseconds after trial onset, we reconstruct the ball's position at any point along its future trajectory with high accuracy (Pearson's $r > 0.8$; Fig. 2E, left). The dSPM model replicates this capability, maintaining high correlation throughout the future trajectory (Fig. 2E, middle). These results further establish that physical prediction in DMFC is a consequence of the geometry of a latent, low-dimensional manifold embedded within the neural activity that encodes a physics-based representation of the gameboard.

Alternative models cannot explain key features of the DMFC activity

To isolate dSPM's explanatory power, we compare it to two distinct classes of alternative models: (i) task-optimized RNNs and (ii) a task-performant heuristic taking advantage of the specifics of the current ball-interception task. First, 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 step 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. Crucially, these models are directly from ref. [26], which showed that the ball interception performance of these models strongly correlates with the behavioral performance of the animals in the current task [26]. Second, we also test a heuristic strategy, called "Linear Map", specifically designed for the current task. The linear map heuristic has access to the moment-by-moment position and velocity of the ball and a linear regression to map this state information to the ball endpoint. Does dSPM provide explanatory power over the DMFC activity, above and beyond what can be explained by these strong alternatives?

Critically, we find that the answer is yes: these alternative models fail to reproduce the full extent of DMFC's future-trajectory-prediction abilities. We summarize these results along three points. First, standard task-optimized RNNs (next-time point and paddle-only models of ref. [26]) fail to reproduce DMFC- and dSPM-like rapid ball endpoint prediction, a result we replicate and extend from Rajalingham et al. [27] (Fig. 2C, right; see Methods). Instead, they show a gradual improvement over time. Moreover, these RNNs qualitatively decouple from neural dynamics at the finer-grained analysis of conditions with only "zero-bounce" or "one-bounce" trajectories (Supplementary Fig. 5).

Second, unlike the DMFC and dSPM, these RNNs also do not recapitulate the full future trajectory prediction, instead showing decreasing prediction accuracy for more distant future time points (Fig. 2E, right). Statistical analysis confirms that when compared to DMFC data, dSPM achieves significantly higher correlation and lower error than these task-optimized RNNs (Fig. 2F).

Third, the simplicity of the gameboard in the current ball interception task allows for a heuristic, non-simulation strategy for predicting the ball's endpoint — the linear map heuristic, which can accurately predict the ball's endpoint from the early position/velocity state vector, similar to DMFC and dSPM (Supplementary Fig. 5). Does the DMFC neural activity employ this specialized heuristic strategy, instead of the structure-preserving nonlinear dynamics that dSPM stipulates? To answer this, we analyzed a divergent prediction made by the linear map heuristic and dSPM (Supplementary Fig. 6): the generalization performance of the ball endpoint decoder across Pong conditions with zero-bounce versus one-bounce trajectories. Linear map yields a time-invariant, nearly perfect generalization of the endpoint decoder across the bounce groups. In contrast, dSPM, due to its bifurcating dynamics for collision detection and resolution, yields a time-dependent and less performant generalization of the endpoint decoder across the bounce groups. When we apply this cross-bounce endpoint decoder analysis to the DMFC activity, we find that it more closely aligns with dSPM, demonstrating a time-dependent cross-bounce generalization. That DMFC qualitatively decouples from the linear map heuristic provides important evidence for dSPM's physics-based representation. Despite the availability of a simpler solution that generalizes perfectly across bounce groups (i.e., the linear map heuristic), DMFC implements condition-specific nonlinear dynamics, much like the bifurcating attractors of dSPM (illustrated in Supplementary Fig. 3 animation).

We suggest that the inability of these alternatives to explain DMFC reflects DMFC's evolution for diverse 3D physical scenes, not just 2D ball tracking on a simple gameboard. Indeed, both DMFC and dSPM match the endpoint decoding performance of the linear map heuristic within each bounce condition (both achieve ~ 0.8 correlation; Supplementary Fig. 5), while implementing rich condition-specific dynamics that does not generalize across bounce conditions. This computational strategy may explain the involvement of frontal circuitry across diverse sensorimotor behaviors requiring physics prediction, including reaching movements [48], pursuit and evasion [49], and object manipulation [50].

Similarly, the failure of task-optimized RNNs to replicate rapid endpoint prediction highlights the central difficulty we identified in the Introduction regarding using standard deep neural networks to study neural algorithms. Despite the rigorous exploration of training objectives and architectures by researchers (e.g., [21, 23, 51]), it is not readily clear what it would take to prevent these task-optimized RNNs from exploiting shortcuts that deliver good task performance without rapid endpoint prediction ability. Training objectives, datasets, and network architectures provide only indirect tools to synthesize the often black-box task-optimized statistical representations in DNNs. In contrast, the dSPM framework empowers researchers with direct explorations of falsifiable algorithmic possibilities of neural mechanisms. Together, these results of alternative models suggest the brain prioritizes physics-based structure-preserving representations over computational shortcuts, even when these learned shortcuts or simple heuristics would suffice for task performance.

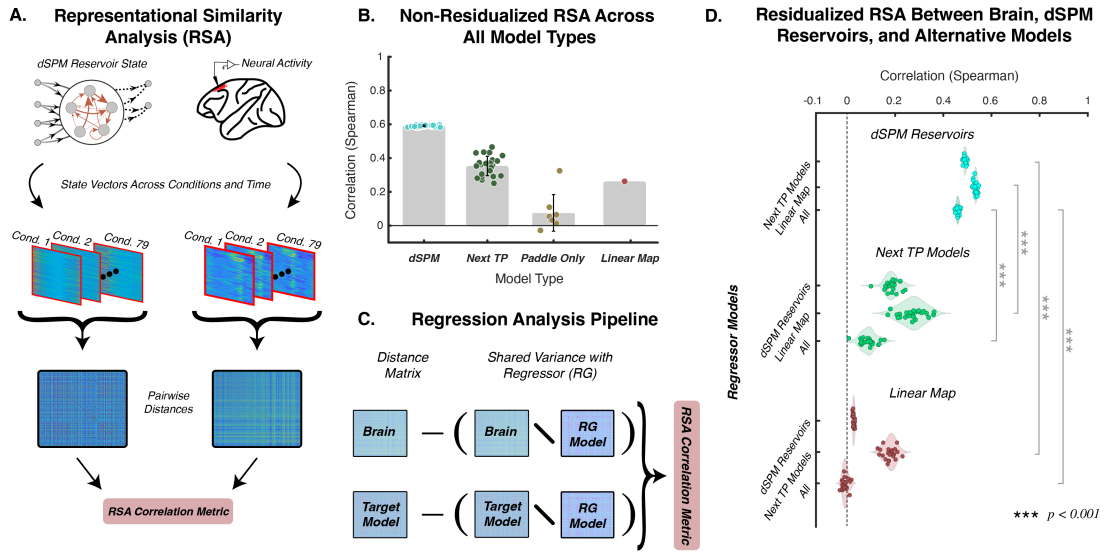


Figure 3. dSPM captures the similarity structure of DMFC neural dynamics across moments and conditions. (A) We performed a representational similarity analysis comparing DMFC neural similarity matrix to the similarity matrices of each model. (B) dSPM reservoirs achieve a significantly higher correlation (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: dSPM; middle: next-time point; bottom: linear map) and residualization condition (controlling for variance explained by different model types). Supplementary Fig. 7 shows the full set of residualization analysis. dSPM shows significantly higher correlations with neural data than alternatives across all residualization settings. Critically, when residualizing dSPM reservoir states from alternatives, there remains little explanatory power of these alternative models, while dSPM maintains a high correlation even after residualizing out all alternative models. This asymmetry indicates that dSPM reservoir captures fundamental aspects of neural computation not present in alternative models.

dSPM captures representational similarity of DMFC dynamics and subsumes what can be explained by alternatives

Finally, we also tested the ability of dSPM and alternative models to explain the representational similarity of neural dynamics across the 79 Pong conditions and moments in these conditions, using representational similarity analysis (RSA; see Methods) [52]. Relative to the decoding-based analysis we have so far focused on, RSA imposes a qualitatively different test of candidate models, involving a direct analysis of their full internal states for explaining DMFC activity. In addition, RSA, via partial correlation, allows us to test whether dSPM maintains explanatory power after accounting for the variance that can be explained by alternative models (the task-performant linear map heuristic and the task-optimized RNNs), which lack the physics-based representation encoded in dSPM.

For a given data source (either a model or DMFC), 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. 3A). We found that the similarity matrix of dSPM achieves significantly and substantially higher correlations with the DMFC similarity matrix ($r = 0.59$), compared to the similarity matrices of the next-time point ($r = 0.35, p < 0.001$) and paddle-only ($r = 0.08, p < 0.001$) models, as well as the linear map heuristic ($r = 0.32, p < 0.001$) (Fig. 3B; all p values indicate pairwise comparisons against dSPM using direct bootstrap hypothesis testing).

We then used partial correlation analysis to ask the extent to which the dSPM explains non-overlapping variance in the DMFC similarity matrix, relative to the task-optimized RNNs and linear map heuristic (Fig. 3C; see Methods). We found that dSPM explains a substantial amount of variance after residualizing both kinds of task-optimized RNNs; but this was not the case for the next-time point model which had nearly no variance left to explain after residualizing the dSPM and paddle-only models (Fig. 3D; see Supplementary Fig. 7 for a full residualization analysis). The differences in the residualized variances explained by dSPM versus next-time point RNN were statistically significant ($p < 0.001$, Fig. 3D, Supplementary Fig. 7).

We found a strikingly similar result when we repeated these partial regression analysis between dSPM and the linear map heuristic (Fig. 3C, D). This heuristic represented each time point for each condition using the ground-truth position and velocity of the ball (corresponding to the "Oracle" covariate in ref. [51], which outperformed all model variants considered in that study). We found that dSPM not only statistically significantly and substantially outperforms this covariate ($r = 0.59$ for dSPM versus $r = 0.32$ for linear heuristic), but also subsumes all of its portion of the explained variance (Fig. 3D). These results strongly suggest a neural mechanism of physical prediction in DMFC through a latent, low-dimensional manifold whose geometry corresponds to a physics-based representation of the gameboard, instead of task-optimized statistical representations or task-performance heuristics.

Discussion

Distinctively, the present work synthesizes two prominent views of the brain, across the fields of cognitive science and neuroscience, which have so far been developed largely independently of each other: The brain as a symbolic representation system of structure-preserving mappings [28, 53] and the brain as a dynamical system of low-dimensional manifolds [17]. Unlike typical cognitive models that focus on computational-level explanations and behavioral data, dSPM penetrates through levels of analysis and offers falsifiable hypotheses of neural mechanisms [54]. And instead of the common neuroscientific approach of analyzing high-dimensional neural data through dimensionality-reduction techniques to visualize low-dimensional manifolds (e.g., [55]), our approach provides a computationally constructive handle on these manifolds. By doing so, this work suggests a neural mechanism of physical prediction in macaque DMFC, as a latent low-dimensional manifold whose geometry corresponds to physics-based representations of scenes.

We focused on neural mechanisms of physical prediction, but we believe dSPM will generalize to other sorts of structure-preserving representations, including representations of agents (e.g., [56]), places [57], and more complex physical scenarios (e.g., [5]). The key to this generalization is to build dynamical algorithms of these domains. (The analytical mapping from dynamical algorithms to reservoir computer is general-purpose.) Our optimism is due to three reasons. First, from the early days of computation to recent times, despite interruptions, dynamical algorithms have been developed for increasingly sophisticated problems, from the "differential analyzer" of Vannevar Bush and colleagues [58] for solving integrals and other engineering problems to virtualization, inverse problems, and dynamical memories by Kim & Bassett [22]. Second, many of the computational primitives we utilized here for representing the gameboard, including objects, if-else branching collision logic, and velocity integration, are common motifs in the structure-preserving representations of more complex physical scenarios and other domains. Third, the strength of evidence we provide for dSPM in the present work motivates us, and hopefully other researchers, to pursue the extensions of dSPM beyond the current case study of physical prediction. For instance, we see the domains of mental navigation as in [47] and online perception, physical prediction, and planning settings as in [49] as immediate next targets for generalizing dSPM.

With such generalization at hand, dSPM promises to be an invaluable reverse-engineering tool 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 to 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 [59, 60]. Similarly, despite the helplessness of human infants, developmental psychology suggests a sophisticated starting point for human cognition [61]. The dSPM model, by synthesizing symbolic representations and dynamical systems, achieves what biology demonstrates: efficient, robust, and interpretable intelligence that begins with, rather than learns, the fundamental rules governing our world.

Methods

Dynamical Structure-Preserving Manifolds (dSPM)

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

Computational-Level Description: Structure-preserving, physics-based representations for physical prediction

At the computational level, physical prediction can be described as building and manipulating physics-based representations of the physical world [29], flexibly supporting downstream adaptive behaviors. Following Battaglia et al. [29], we provide a structure-preserving physics-based representation of the gameboard 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. The dSPM framework transforms such computational-level hypothesis, which lack neural grounding, into falsifiable proposals of neural mechanisms.

Algorithmic-Level: Dynamical Algorithm

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. To realize complex cognitive representations, we need to express significantly more sophisticated computations in these dynamics.

To do so, we build on Kim & Bassett [42] to formulate a dynamical algorithm of a physics-based representation of the gameboard. This dynamical algorithm (Supplementary Fig. 2, 3) is a set of 12 coupled differential equations whose outputs are denoted as $\{z_1, z_2, \ldots, z_{12}\}$. This dynamical algorithm defines a dynamical manifold, meaning that it's a manifold whose attractors and repellers change over time. These dynamical attractors — i.e., nullclines in phase space that change with time t — correspond to a physics-based representation of the gameboard, including position updates, as well as collision detection and resolution. 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, [62])

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

- **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) [63], 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 (Supplementary Fig. 2), making slight changes in the system more sensitive to changes. (This time constant is multiplicative to the global time constant of our reservoir computer, represented by γ in our update equation in the next section.)

The four computational blocks of our dynamical algorithm 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:

$$|v_x| \mapsto \dot{z}_1 = 0 \qquad (\dot{z}_1)$$
$$|v_y| \mapsto \dot{z}_2 = 0 \qquad (\dot{z}_2)$$

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

2. Velocity Logic

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

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: A Reservoir Computer Analytically Embedding the Dynamical Algorithm

In contrast to the traditional task-optimization approach that learns the weights via numerical training in a deep neural network, we constructed a separate class of models, i.e., reservoir computers, by analytically embedding the dynamical algorithm defined above (the twelve differential equations encoding ball motion, wall collision logic, and velocity sign changes) into the reservoir computer's connectivity weights. We do so by following the method of Kim & Bassett [42].

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. [42])

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^*

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

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 Reservoir Computer".

Applying dSPM to Experimental Conditions

Once we have our programmed reservoir computer, we can "load in" the initial conditions and evolve our network over time. As the reservoir evolves its distributed computation, across the N hidden units, together computes the dynamical algorithm that constitute our physics-based representation of the gameboard.

We initialize dSPM with the initial conditions used within the monkey experiments. The initial conditions contain ball position and velocity for each trial, which dSPM evolves autonomously without external inputs for the duration of the given Pong condition.

Traditional Task-Optimized Deep Neural Networks

Rajalingham et al. [26] trained a large ensemble of standard machine-learning-style RNNs on the same interception task, using 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.

The next time-point models were trained to predict 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 different variants of the next time-point models depending on the specifics of the loss and architecture, which we pool together as they did not differ from each other statistically. The paddle-only models were trained only the latter objective — the ball endpoint.

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 dSPM.

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. [26, 27]. 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 $\times$ 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) [46] fits to decode the positions of the ball from the latent state-vectors across time. Neural population responses, task-optimized RNN hidden states, and dSPM reservoir computer states were all compared via two complementary analyses: ball endpoint decoding (Fig. 2) and full trajectory decoding (Fig. 2). All GLM results reported are cross-validated using nested 5-fold cross-validation with 4-fold inner cross-validation for hyperparameter selection.

For ball endpoint 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 Fig. 2E 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 $\times$ 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.

Representational 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 dSPM 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. Residualization was performed as the following: We randomly sampled a single model instance from the source model class as the regressor, repeated across all model instances to account for sampling variability. This procedure was applied bidirectionally, residualizing dSPM models from task-optimized models and vice versa. In Fig. 3D, because the linear map heuristic has no variability (as it is based on the ground truth), our method carries over the variability due to dSPM instances for statistical quantification. We report approach all possible residualization pairings in Supplementary Fig. 7. Statistical significance was determined using parametric pairwise comparisons between residualized correlation values (paired samples two-tailed t-test). [Notice that we have equal number (24) of dSPM instances and task-optimized RNNs.] The asymmetric pattern of residualization results — where dSPM maintained high correlation with neural data after removing task-optimized and linear heuristic's variance, but not vice versa—indicates that dSPM capture fundamental aspects of neural computation not present in the alternative models.

Computational Implementation

All 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 model analyses, and all time-series were binned at 50 ms. Optimal hyperparameters in regression or decoding analyses (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). This work was supported by National Science Foundation (under CAREER Award No. BCS-2441520).

References

1. Thomas L Griffiths, Nick Chater, Charles Kemp, Amy Perfors, and Joshua B Tenenbaum. Probabilistic models of cognition: exploring representations and inductive biases. *Trends Cogn. Sci.*, 14(8):357–364, August 2010.

2. C R Gallistel and Adam Philip King. *Memory and the Computational Brain: Why Cognitive Science will Transform Neuroscience*. John Wiley & Sons, September 2011.

3. 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*, 110(45):18327–18332, 2013.

4. 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.

5. 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.

6. Thomas L Griffiths, Edward Vul, and Adam N Sanborn. Bridging levels of analysis for probabilistic models of cognition. *Curr. Dir. Psychol. Sci.*, 21(4):263–268, August 2012.

7. David Marr. Vision: A computational approach, 1982.

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

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

10. 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.

11. Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, 12 2013.

12. Blake A Richards, Timothy P Lillicrap, Philippe Beaudoin, Yoshua Bengio, Rafal Bogacz, Amelia Christensen, Claudia Clopath, Rui Ponte Costa, Archy de Berker, Surya Ganguli, Colleen J Gillon, Danijar Hafner, Adam Kepecs, Nikolaus Kriegeskorte, Peter Latham, Grace W Lindsay, Kenneth D Miller, Richard Naud, Christopher C Pack, Panayiota Poirazi, Pieter Roelfsema, João Sacramento, Andrew Saxe, Benjamin Scellier, Anna C Schapiro, Walter Senn, Greg Wayne, Daniel Yamins, Friedemann Zenke, Joel Zylberberg, Denis Therien, and Konrad P Kording. A deep learning framework for neuroscience. *Nat. Neurosci.*, 22(11):1761–1770, November 2019.

13. Li Ji-An, Marcus K Benna, and Marcelo G Mattar. Discovering cognitive strategies with tiny recurrent neural networks. *Nature*, pages 1–9, 2025.

14. David Sussillo and Omri Barak. Opening the black box: low-dimensional dynamics in high-dimensional recurrent neural networks. *Neural Comput.*, 25(3):626–649, March 2013.

15. R Shepard and S Chipman. Second-order isomorphism of internal representations: Shapes of states . *Cognitive Psychology*, 1:1–17, 1970.

16. Christian K Machens, Ranulfo Romo, and Carlos D Brody. Flexible control of mutual inhibition: a neural model of two-interval discrimination. *Science*, 307(5712):1121–1124, February 2005.

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

18. Jimmy Smith, Scott Linderman, and David Sussillo. Reverse engineering recurrent neural networks with jacobian switching linear dynamical systems. In M Ranzato, A Beygelzimer, Y Dauphin, P S Liang, and J Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 16700–16713. Curran Associates, Inc., 2021.

19. Victor Geadah, International Brain Laboratory, and Jonathan W Pillow. Parsing neural dynamics with infinite recurrent switching linear dynamical systems. *Int Conf Learn Represent*, 2024.

20. 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.

21. Rishi Rajalingham, Hansem Sohn, and Mehrdad Jazayeri. Dynamic tracking of objects in the macaque dorsomedial frontal cortex. *Nat. Commun.*, 16(1):346, January 2025.

22. Jason Z Kim and Dani S Bassett. A neural programming language for the reservoir computer. *arXiv [cond-mat.dis-nn]*, March 2022.

23. Rishi Rajalingham, Aída Piccato, and Mehrdad Jazayeri. Recurrent neural networks with explicit representation of dynamic latent variables can mimic behavioral patterns in a physical inference task. *Nat. Commun.*, 13(1):5865, October 2022.

24. 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.

25. Hamed Nili, Cai Wingfield, Alexander Walther, Li Su, William Marslen-Wilson, and Nikolaus Kriegeskorte. A toolbox for representational similarity analysis. *PLoS Comput. Biol.*, 10(4):e1003553, 2014.

26. 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.

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

28. C. R. Gallistel and Adam Philip King. *Memory and the Computational Brain*. 2009.

29. 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.

30. 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.

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

32. 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.

33. 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.

34. Kevin Smith, Lingjie Mei, Shunyu Yao, Jiajun Wu, Elizabeth Spelke, Josh Tenenbaum, and Tomer Ullman. Modeling expectation violation in intuitive physics with coarse probabilistic object representations. *Adv. Neural Inf. Process. Syst.*, 32, 2019.

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

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

37. 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.

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

39. J J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proc. Natl. Acad. Sci. U. S. A.*, 79(8):2554–2558, April 1982.

40. J J Hopfield and D W Tank. Computing with neural circuits: a model. *Science*, 233(4764):625–633, August 1986.

41. Vishwa Goudar and Dean V Buonomano. Encoding sensory and motor patterns as time-invariant trajectories in recurrent neural networks. *Elife*, 7:e31134, March 2018.

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

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

44. Steven Strogatz, Mark Friedman, A John Mallinckrodt, and Susan McKay. Nonlinear dynamics and chaos: With applications to physics, biology, chemistry, and engineering. *Comput. Phys. Commun.*, 8(5):532, 1994.

45. Subhodh Vyas, Matthew D Golub, David Sussillo, and Krishna V Shenoy. Computation through neural population dynamics. *Annual Review of Neuroscience*, 43:249–275, 2020.

46. 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.

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

48. 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.

49. 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.

50. 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.

51. 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.

52. 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.

53. 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.

54. 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.

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

56. Chris L Baker, Rebecca Saxe, and Joshua B Tenenbaum. Action understanding as inverse planning. *Cognition*, 113(3):329–349, December 2009.

57. Russell A Epstein, Eva Zita Patai, Joshua B Julian, and Hugo J Spiers. The cognitive map in humans: spatial navigation and beyond. *Nat. Neurosci.*, 20(11):1504–1513, October 2017.

58. V Bush, F D Gage, and H R Stewart. A continuous integraph. *J. Franklin Inst.*, 203(1):63–84, January 1927.

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

60. 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.

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

62. 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.

63. 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.

Supplementary Material

Details of Programming Reservoir Computer

In brief, the framework for programming the weights and connectivity of a reservoir computer [42] is as follows:

1. Define the Network Update Equation

Here, we define our to-be-programmed Reservoir Computer 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 reservoir 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 reservoir at time t
4. $A \in \mathbb{R}^{N\times N}$, is the adjacency matrix describing the connectivity of the reservoir units.
5. $B \in \mathbb{R}^{N\times M}$, is the "read-in" matrix describing the exogenous connectivity into the reservoir.
6. $\vec{d} \in \mathbb{R}^{N\times 1}$, is the constant bias vector for each unit in the reservoir.

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 reservoir 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 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 Dynamical Variables

We next define a set of dynamical variables $\vec{z}(t) \in \mathbb{R}^{P\times 1}$ that encode the desired physics-based representation. Each dynamical 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. 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 The Resulting Dynamical Algorithm

Given our dynamical algorithm (i.e., the 12 dynamical variables $\dot{\vec{z}} = f(\vec{z})$), 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 dynamical variable expressed in the same basis as our reservoir expansion.

6. Program the Readout Matrix

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

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

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 reservoir computer:

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

where $A^* = \hat{B}W$ is the effective connectivity matrix encoding our dynamical algorithm.

This framework enables the analytical embedding of arbitrary dynamical systems into reservoir computer, transforming differential equations into distributed connectivity patterns that autonomously execute the desired computations. The reader can consult ref. [42] 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 reservoir computer 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

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
```

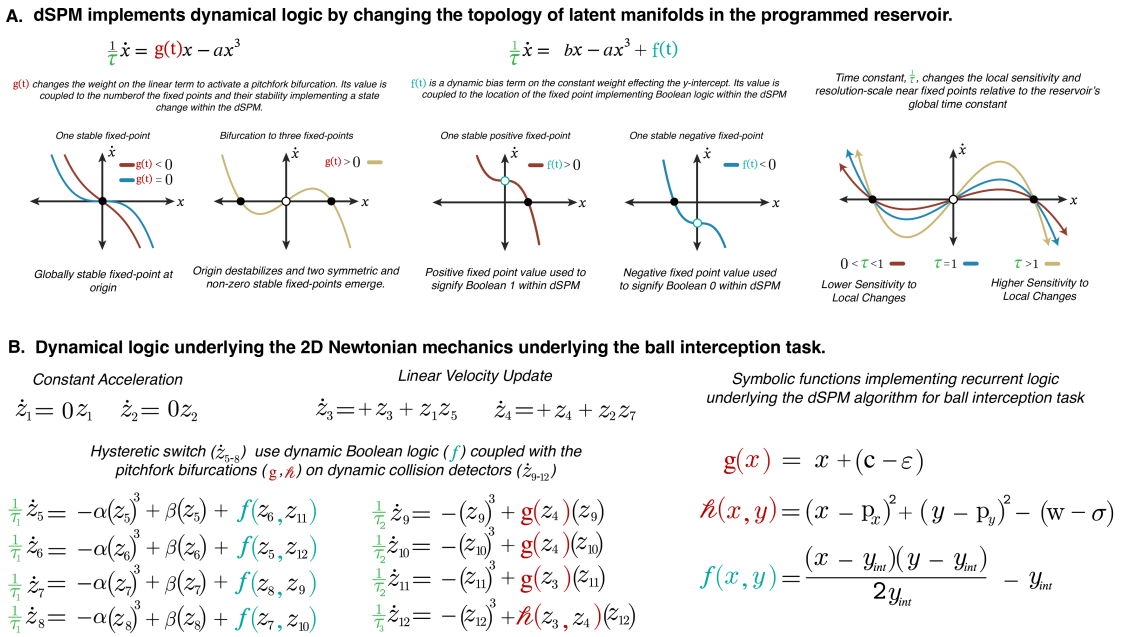
Non-linear Velocity Flip if Collision is Detected

```
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
```

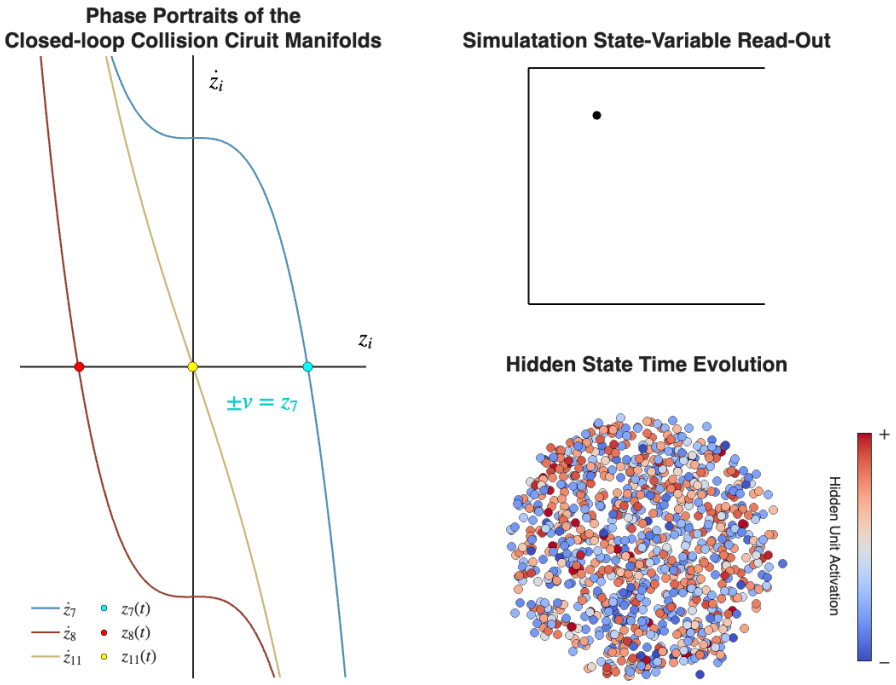
Linear Position Update Using Hysteretic Velocity State

```
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
```

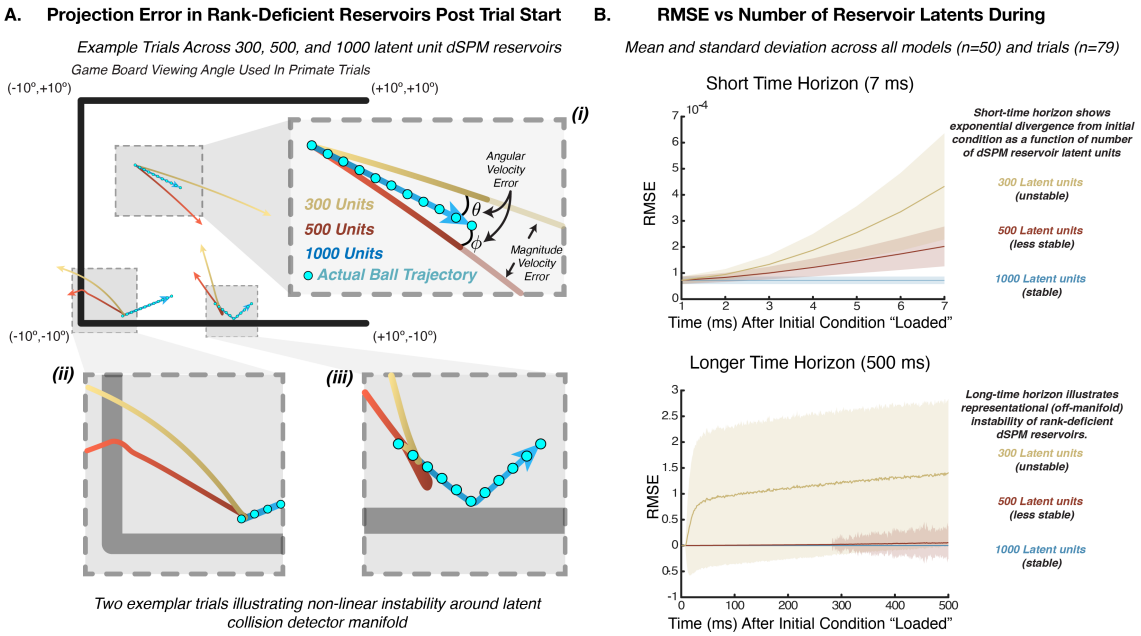
Supplementary Figure 1. Pseudocode of the physics-based representation of the gameboard, which underlies our dSPM construction. This figure presents an intuitive pseudocode representation to illustrate how the 12 coupled differential equations (Equations ($\dot{z}_1$) through ($\dot{z}_{12}$)) described in the Methods. (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 dSPM, this physics-based representation is analytically embedded into the connectivity matrix of a reservoir computer, $A = \hat{B}W$.



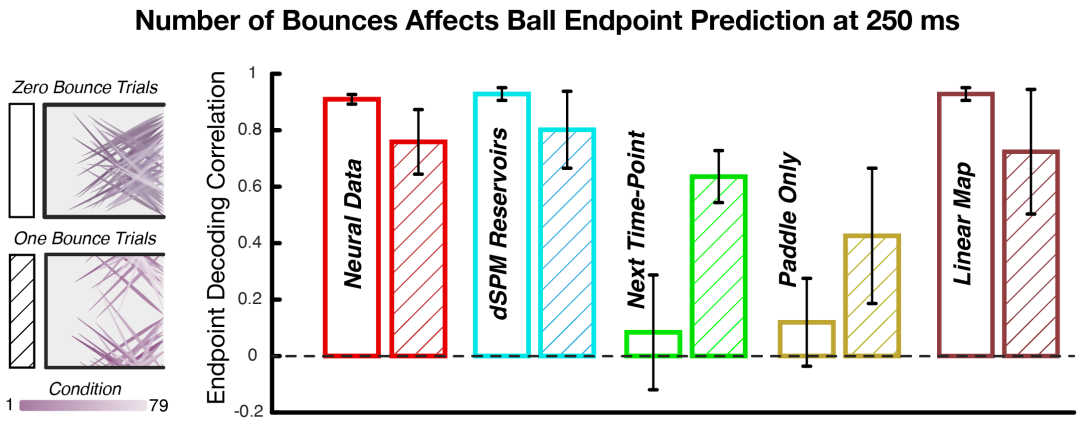
Supplementary Figure 2. Example dynamical primitives for implementing symbolic representations. (A) Left two panels show the phase portraits of a controlled bifurcation with the equation $\frac{1}{\tau}\dot{x} = g(t)x - ax^3$. The stable fixed point (attractor; filled circle) on the left transitions into two stable fixed points and one unstable fixed points (repeller; unfilled circle) as $g(t)$ becomes positive. We use this type of bifurcation for collision detection in our gameboard representation. Middle two panels show the same type of bifurcation but this time with its constant term time-varying, $\frac{1}{\tau}\dot{x} = bx - ax^3 + f(t)$. The stable and unstable fixed points non-linearly transition as a function of the time-varying constant term $f(t)$. In our gameboard representation, we use this type of bifurcation for the velocity sign variable. Through these time varying coefficients (e.g., $g(t)$ and $f(t)$), we can couple different dynamical variables and implement branching logic, linear velocity dynamics, and other elements needed to represent the physical scenes and other structure-preserving representations. The rightmost panel shows how the sensitivity of these dynamical primitives can be changed by adjusting the value of τ. This rightmost panel is plotting the second plot from left (i.e., $\frac{1}{\tau}\dot{x} = g(t)x - ax^3$ for $g(t) > 0$) with different values of the time constant. (B) The full dynamical algorithm, consisting of the 12 dynamical variables, as described in Methods.



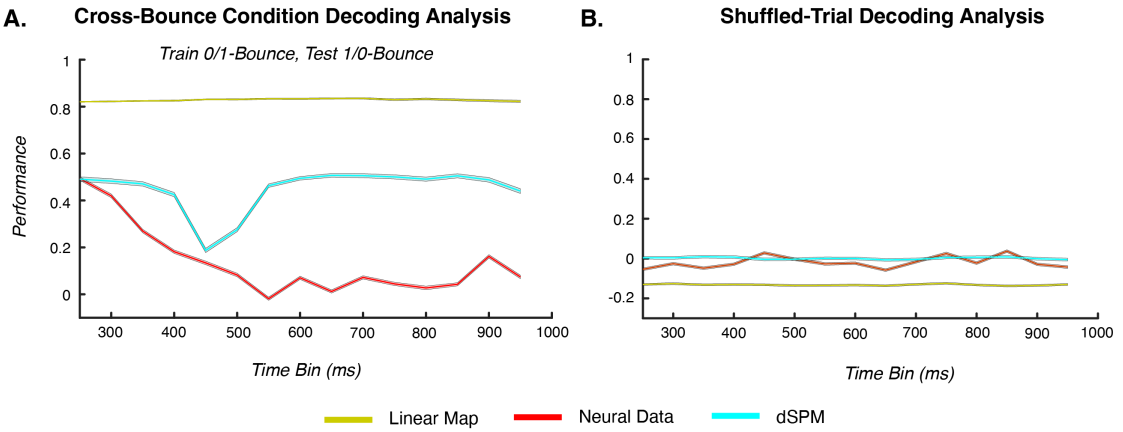
Supplementary Figure 3. Animated visualization of the dSPM dynamical algorithm and reservoir state evolution (1000 units) implementing non-linear wall collision detection and resolution. Left panel: Phase portraits of the collision circuit's recurrent dynamics, showing the theoretical nullclines (continuous 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. (NOTE: To play the animation within the PDF, please use Adobe Acrobat. This animation is also included as a Supplementary Video.)



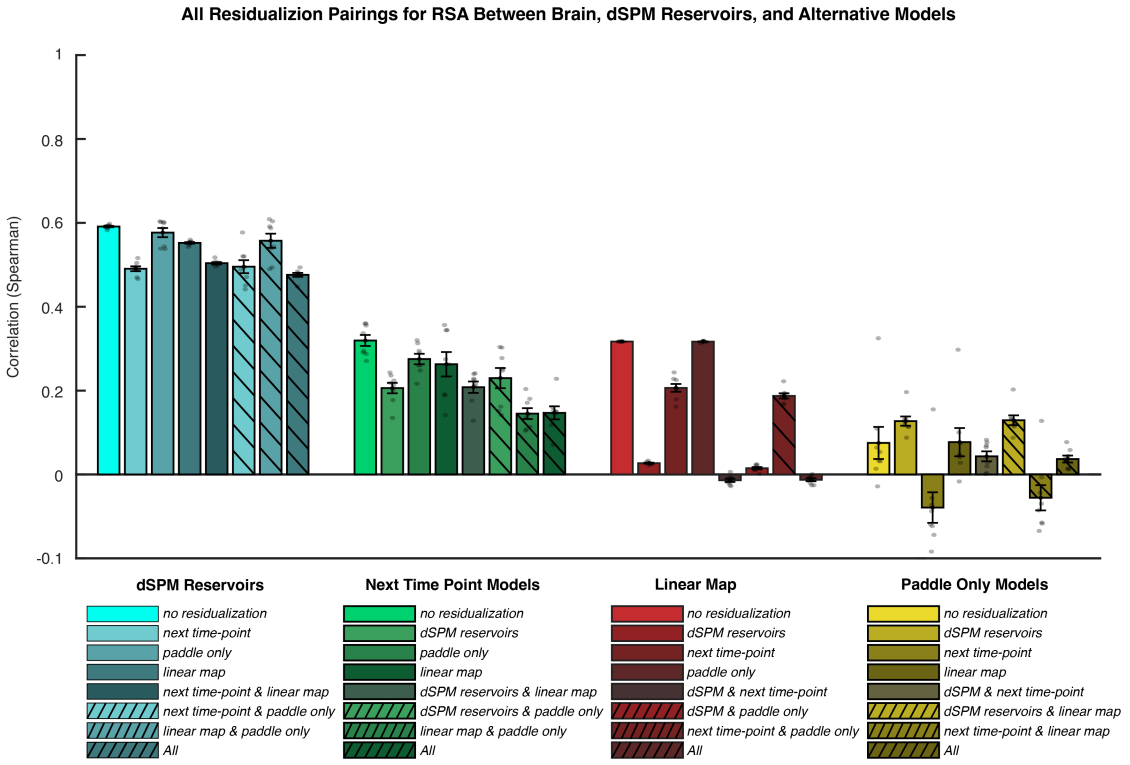
Supplementary Figure 4. Stability analysis of dSPM with varying numbers of hidden units. (A) Projection error visualization in networks after trial start, showing example trajectories from dSPM reservoir computer with 300 (yellow), 500 (red), and 1000 (blue) hidden units. The main panel displays ball 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-iii) 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 dSPM size during the initial trial phase, averaged across 50 models and 79 conditions per model size. Short time horizon (7 ms; top panel) 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. Longer time horizon (500 ms; bottom panel) 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 5. Effect of trajectory complexity on rapid (at 250 milliseconds) ball endpoint prediction. DMFC populations show highly accurate ball endpoint prediction for both within zero-bounce and within one-bounce trials, with slightly better performance on the less complex zero-bounce trials. dSPM recapitulates DMFC-like robust prediction for both condition types. Task-optimized RNNs show generally poor performance, whereas the Linear Map also recovers DMFC-like pattern.



Supplementary Figure 6. Heuristic strategies versus dynamical computation in DMFC. (A) To distinguish between the Linear Map heuristic and dSPM'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 dSPM and DMFC. All of these decoders are trained on a given time point (along the x-axis) to predict the ball endpoint. 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 dSPM (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 dSPM and DMFC: In dSPM, 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 endpoint decoding is due to some spurious correlations in the dataset. To rule out this possibility, we train and test ball endpoint 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 7. The full set of RSA residualization results. The dSPM model explains more variance than other models and most of this variance is orthogonal to what other models can explain. The reverse is not true: dSPM, for the most part, subsumes what can be explained by the task-optimized RNNs and the task-performant linear map heuristic. (For the paddle-only model, because it's essentially uncorrlated from neural data, our partial regression analysis "pumps in information", leading to the counterintuitive pattern of increased variance after residualization.)