Discovering Mechanistic Models of Neural Activity: System Identification in an *in Silico* Zebrafish

Jan-Matthis Lueckmann
Google Research
janmatthis@google.com

Viren Jain
Google Research
viren@google.com

Michał Januszewski
Google Research
mjanusz@google.com

Abstract

Constructing mechanistic models of neural circuits is a fundamental goal of neuroscience, yet verifying such models is limited by the lack of ground truth. To rigorously test model discovery, we establish an in silico testbed using neuromechanical simulations of a larval zebrafish as a transparent ground truth. We find that LLM-based tree search autonomously discovers predictive models that significantly outperform established forecasting baselines. However, in-distribution accuracy is a poor proxy for faithful system identification, as models exploit statistical shortcuts. Structural priors prove essential for enabling robust out-of-distribution generalization and recovery of interpretable mechanistic models. Our insights provide guidance for modeling real-world neural recordings and offer a broader template for AI-driven scientific discovery.

1 Introduction

A fundamental goal in neuroscience is to understand how neural circuits process information and drive behavior. The ability to predict neural activity is a key test towards such understanding. Data-driven benchmarks can accelerate progress by providing standardized datasets and evaluation metrics [1–4].

However, benchmarks relying on real experimental data face a fundamental verification gap: because the ground-truth data-generating process is unknown, models are ranked relative to each other based on predictive performance rather than compared against an absolute reference. It is thus unclear whether a highly predictive model is actually capturing the true mechanisms of computation, or whether it has merely identified *statistical shortcuts* (a glossary of *key terms* is in App. A). This inability to verify *mechanistic recovery* is compounded by limitations of biological experiments. For instance, limitations include the high cost and technical difficulty of acquiring large-scale, high-resolution neural recordings, the presence of noise and unobserved variables, and restrictions in systematically manipulating neural circuits or environments. Real-world experiments that involve imaging of neural activity are severely restricted in duration and throughput (e.g., due to phototoxicity, molecular reporters, and imaging limitations), and subject to artifacts and preprocessing challenges (e.g., motion, striping, imperfect alignment and segmentation). Without a known ground truth, it is thus hard to rigorously evaluate models and model discovery strategies.

Neuromechanical simulations provide a powerful avenue to complement real-data benchmarks: Embodied simulations can generate vast amounts of perfectly annotated data where the underlying "ground truth" of the neural system is known and controllable. They allow for systematic manipulation of physiological and circuit features that are difficult or impossible to achieve in biological experiments—offering full observability and control over factors such as dataset dimensionality, length, and noise levels, along with the ability to perturb any part of their neural circuitry. simZFish [5], an open-source neuromechanical simulation of the larval zebrafish, is one such platform that realistically models the fish's body, its physical interactions with the environment, visual inputs, and experimentally constrained neural architectures for visuomotor behaviors.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

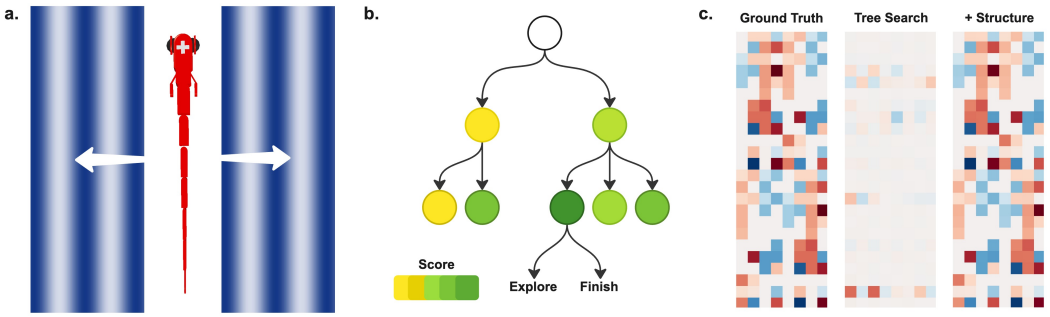

Figure 1: **Verifiable discovery of neural mechanisms in an in silico testbed. a.** The simulation environment consists of a neuromechanical model subject to fluid dynamics, responding to visual stimuli driven by a neural circuit in a closed-loop setting. **b.** We use LLM-guided tree search to autonomously explore the space of dynamical models, evolving Python code to minimize predictive error on neural activity. **c.** Despite high predictive in-distribution performance, unconstrained black-box tree search models fail to identify the system's mechanisms, as revealed by effective connectivity matrices (excerpt shown; blue and red indicate inhibitory and excitatory interactions, respectively; color intensity represents magnitude). In contrast, a structure-constrained grey-box tree search model successfully identifies the correct signs and magnitudes closely matching the ground truth, from a structural prior that provides information about the existence and absence of connections.

In this work, we use simZFish to generate high-fidelity in silico datasets that serve as a testbed for *system identification* [6]. By simulating the virtual fish's response to a range of visual stimuli, we create a ground-truth dataset where every internal state, sensory input, and underlying circuit connectivity is perfectly observable. This allows us to study a critical question: What are the data and modeling prerequisites to recover the mechanisms of a complex neural circuit?

While standard forecasting architectures can provide initial baselines, they are rarely optimized for the specific temporal and structural constraints of neural dynamics. A primary bottleneck in modeling neural activity is often the modeler's own inductive bias and the significant time required to manually explore the architectural landscape. We address this by framing model design as an optimization problem. Specifically, we utilize an automated tree-search approach [7] to traverse a vast space of code-based model definitions, leveraging Large Language Models (LLMs) to effectively "evolve" solutions. We compare two distinct discovery regimes: an unconstrained search for high-accuracy black-box predictors, and a structure-constrained search that uses *structural priors* (derived from the "wiring diagram"), and we evaluate the discovered models with respect to their ability to generalize to unseen stimuli, their interpretability, and mechanistic recovery (Fig. 1).

In summary, our contributions are as follows:

1. **An in silico testbed for verifiable neural system identification.** Unlike real-data benchmarks where ground truth is unknown, our testbed provides a transparent reference, allowing us to evaluate whether models recover the true mechanisms of a neural circuit.
2. **Sensory drive is a prerequisite for identifiability.** We show that the true transition function is *non-identifiable* using autoregressive history alone, with implications for real-world benchmarks.
3. **In-distribution accuracy is a poor proxy for system identification.** We show that LLM-guided tree search discovers models that achieve state-of-the-art test set accuracy but fail on out-of-distribution (OOD) stimuli. These models prioritize statistical shortcuts over faithful recovery of mechanisms, challenging the practice of ranking neural models by test set error.
4. **Structural priors enable mechanistic recovery.** We demonstrate that providing structural priors is sufficient to regularize the discovery process. This eliminates statistical shortcuts, enables robust OOD generalization, interpretability, and faithful recovery of static and dynamic mechanisms—as verified by *effective connectivity* and impulse response analyses, respectively.
5. **Recommendations for real-world system identification:** Based on our in silico results, we propose concrete guidelines for modeling real neural recordings. We recommend shifting from pure autoregression to connectome-constrained conditional forecasting tasks with OOD evaluations, to bridge the gap between predictive AI and mechanistic neuroscience.

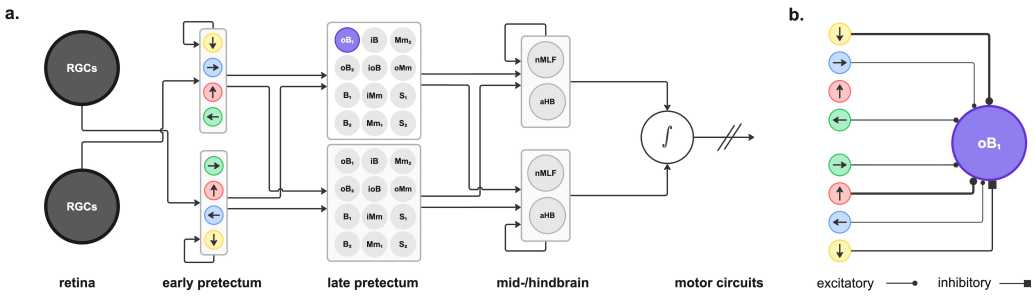

Figure 2: **Neural circuit model. a.** The neural circuit model defines the information flow, processing retinal input through the early and late pretectum (ePT, lPT) to drive command nuclei (nMLF, aHB) and motor circuits. Arrows inside circles represent cardinal direction tuning of ePT neurons to visual motion. **b.** Connectivity diagram of an example late pretectal neuron (oB_1), illustrating the specific excitatory and inhibitory wiring with ePT neurons resulting in its direction-selectivity.

2 Related Work

Automated Model Discovery. Our work uses the capability of Large Language Models (LLMs) to perform automated model discovery via code generation. While traditional approaches to model discovery relied on genetic programming [8] or reinforcement learning for neural architecture search [9], recent methods treat the search process as an evolution of executable code. We utilize the tree-search methodology proposed by Aygün et al. [7]. Parallel efforts have explored similar LLM-driven evolutionary strategies for mathematical discovery [10] and algorithmic evolution [11–13]. In related domains, automated model discovery techniques have for example been applied to search for interpretable cognitive models of human and animal behavior [14], and neural tuning curves [15]. Our work distinguishes itself by applying these methods to neural system identification in a setting where we can rigorously benchmark the discovery of mechanisms against a verifiable ground truth.

System Identification with Ground Truth. The difficulty of inferring ground-truth mechanisms from neural data is a long-standing methodological concern. Jonas and Kording [16] utilized a microprocessor to demonstrate that standard data analysis techniques can fail to recover the known logic of information processing. More recently, Han et al. [17] used artificial neural networks (ANNs) as in silico ground truths, revealing that functional similarity (predictive performance) does not imply mechanistic similarity. We advance this line of inquiry by moving beyond artificial proxies—such as microprocessors or generic ANNs—which differ fundamentally from biological substrates. Instead, we employ a high-fidelity, experimentally constrained emulation of the zebrafish optomotor response to move towards more realistic models with ground truth.

Neuromechanical Simulations and Connectome Constraints. Our work is situated within the growing field of neuromechanical modeling. While detailed simulations have been established for invertebrates such as the worm *C. elegans* [18, 19] and the fruit fly *D. melanogaster* [20, 21], we utilize a larval zebrafish simulation to capture vertebrate visuomotor dynamics. Concurrently, the increasing availability of connectomes has spurred research into constraining neural models with physical wiring diagrams. The success of our structurally constrained models aligns with the findings of Sourmpis et al. [22], who demonstrated that biologically informed models are more robust to optogenetic perturbations than generic RNNs. Several studies, including Mi et al. [23], Achterberg et al. [24], Lappalainen et al. [25], Beiran and Litwin-Kumar [26], Duan et al. [27], have highlighted the predictive utility of structural constraints. Our work bridges these findings with automated discovery: rather than manually designing constrained architectures, we show that connectomic information can be used with AI-driven search to navigate the vast space of possible models and recover mechanistic solutions.

3 Methods

3.1 Neuromechanical Simulations

We generate synthetic neural activity datasets with simZFish, a detailed neuromechanical simulator of larval zebrafish [5]. It integrates realistic body mechanics, environmental interactions, and neural

circuits to reproduce sensorimotor behaviors like the optomotor response (OMR). The OMR is an innate reflex causing fish to orient in the direction of whole-field visual motion to stabilize position, and it is probed in openly available whole-brain activity recordings [e.g., 28, 4]. We provide a summary of the simulations, and refer to Liu et al. [5] and App. B for full details.

Simulation Environment. Simulations are implemented in Webots [29], an open-source robotics simulator. An illustration of the simulation environment is in Fig. 1a.

Body Model. The body approximates a 4 mm larval zebrafish with density slightly less than water. It comprises seven cuboid segments—a head with cameras and pectoral fins, followed by six body segments, and a tail fin—connected by servomotor-actuated hinge joints.

Sensory Environment. Visual stimuli (drifting sinusoidal gratings) are presented in a simulated petri dish, locked to the body orientation, mimicking closed-loop setups.

Fluid Dynamics. To simulate realistic locomotion, the body is subject to buoyancy, viscous resistance, and inertial drag, with parameters tuned to match kinetic recordings.

Our simulations employ a 0.05 ms integration time step using Webots' default first-order semi-implicit integrator for the body kinematics.

Neural Circuit Model. The neural circuit model is based on anatomical, functional, and computational findings from real zebrafish visuomotor pathways [30]. We ported the C-based implementation by Liu et al. [5] to JAX [31], for differentiability and ease of experimentation. To summarize the core mechanisms, we restate the model below using a more compact mathematical notation; these equations do not represent new modeling work. An overview of the network model is in Fig. 2.

The circuit is formulated as a dynamical system where the activity of a neural population is determined by a non-linear transition operator. We define a generalized sigmoidal response function, Φ, parameterized by synaptic weights $\mathbf{W}$, bias thresholds $\mathbf{b}$, and response gains $\boldsymbol{\omega}$:

$$\Phi(\mathbf{x}; \mathbf{W}, \mathbf{b}, \boldsymbol{\omega}) = \sigma\left(\boldsymbol{\omega} \odot (\mathbf{W}\mathbf{x} - \mathbf{b})\right), \tag{1}$$

where $\mathbf{x}$ is the input vector, $\sigma(\cdot)$ is the element-wise sigmoid function, and $\odot$ denotes the Hadamard product.

Retina. Visual input is captured by two laterally positioned simulated cameras, acting as photoreceptors driving a layer of simulated Bipolar Cells (BCs) detecting transient luminance changes. BCs project to four types of Direction-Selective Retinal Ganglion Cells (DSGCs), tuned to the four cardinal directions, producing an output vector $\mathbf{u}_t^{\text{ret}}$ representing the instantaneous motion across the visual field.

Early Pretectum. The retinal output projects contralaterally to the Early Pretectum (ePT). Unlike the retina, pretectal responses exhibit significant temporal persistence. The ePT state $\mathbf{h}_t^{\text{ept}}$ is defined as a first-order low-pass filter driven by sparse retinotopic projections:

$$\mathbf{h}_t^{\text{ept}} = (1 - \alpha_t)\mathbf{h}_{t-1}^{\text{ept}} + \alpha_t \Phi(\mathbf{u}_t^{\text{ret}}; \mathbf{W}_{\text{ret}}, \mathbf{b}_{\text{ept}}, \boldsymbol{\omega}_{\text{ept}}), \tag{2}$$

where $\mathbf{W}_{\text{ret}}$ is a fixed, sparse binary mask enforcing contralateral connectivity to the lower-temporal visual field. Crucially, the update rate α_t is state-dependent: its value depends on the current motor state (swim bout vs. rest) to dynamically filter out self-generated visual motion artifacts during rapid tail beats.

Late Pretectum. The ePT neurons drive the Late Pretectum (lPT), a layer responsible for synthesizing complex binocular motion primitives. This layer consists of 12 distinct functional types per hemisphere, whose names reflect three axes of variation: (1) direction (inward/outward/global); (2) ocularity (monocular/binocular); and (3) velocity preference (type 1/type 2; forward- vs. backward-preferring). For example, an oB_1 neuron responds to outward binocular motion and prefers forward velocity.

Mathematically, we model the lPT as an instantaneous layer combining linear connectivity with a non-linear binocular gating mechanism. We express this compactly by treating the gating signal as a state-dependent threshold modulation:

$$\mathbf{h}_t^{\text{lpt}} = \Phi\left(\mathbf{h}_t^{\text{ept}}; \mathbf{W}_{\text{lpt}}, \mathbf{b}_{\text{lpt}} - \mathbf{M}_{\text{gate}} \odot \mathbf{g}(\mathbf{h}_t^{\text{ept}}), \boldsymbol{\omega}_{\text{lpt}}\right). \tag{3}$$

Here, $\mathbf{W}_{\text{lpt}}$ encodes the subtype-specific excitatory/inhibitory wiring (e.g., Fig. 2b). The term $\mathbf{g}(\cdot)$ represents a binocular co-activation signal (AND-gate) implemented via a minimum function that

selects specific pairs of ipsilateral and contralateral inputs (see App. B for details). This modulation effectively acts as a dynamic bias, controlled by the sparse mask $\mathbf{M}_{\text{gate}}$ (active only for two lPT neurons per hemisphere).

Command Neurons in Mid- and Hindbrain (CMD). The lPT layer projects to two downstream command centers: the Nucleus of the Medial Longitudinal Fasciculus (nMLF) controls swim vigor, and the Anterior Hindbrain (aHB), controlling turning bias. They are modeled as first-order low-pass filters with timescales β and connectivity:

$$\mathbf{h}_t^{\text{cmd}} = (1-\beta)\mathbf{h}_{t-1}^{\text{cmd}} + \beta\Phi(\mathbf{h}_t^{\text{lpt}}; \mathbf{W}_{\text{cmd}}, \mathbf{b}_{\text{cmd}}, \boldsymbol{\omega}_{\text{cmd}}). \tag{4}$$

While the nMLF integrates inputs to drive forward velocity, aHB integrates bilateral differences for the turning direction.

Motor Circuits. Locomotion is discrete (burst-and-glide): a stochastic integrate-and-fire mechanism accumulates drive from the nMLF to trigger swim bouts, whose direction is determined by the state of command neurons. Spinal central pattern generators then produce the rhythmic tail undulations (details and equations in App. B).

Simulated Behavior. The simulated fish executes discrete bouts consisting of rapid tail undulations followed by a passive glide phase, similar to real zebrafish. When exposed to drifting sinusoidal gratings, it recapitulates the behavioral distribution of swims and turns observed in real zebrafish. Crucially, the full embodied simulation ensures this behavior naturally alters the visual input in a closed loop, generating realistic sensorimotor correlations rather than relying on hand-prescribed statistics.

Dataset Generation. We ran simZFish simulations under four binocular motion conditions designed to probe the visuomotor circuit: bilateral outward, bilateral inward, and two asymmetric configurations where eyes receive opposing inward/outward motion. At the same time, we recorded the activity, resulting in 400,000 samples at 1 ms resolution in total.

3.2 System Identification

We propose a system identification [6] task using this dataset in order to establish a rigorous testbed for verifiable model discovery.

Task Definition. We assume neural activity evolves according to a ground-truth transition function f^* driven by past activity and sensory information:

$$\mathbf{a}_t = f^*(\mathbf{a}_{t-1}, \mathbf{x}_t), \tag{5}$$

where $\mathbf{a}_t \in \mathbb{R}^N$ is the activity of the $N = 36$ neural units (comprised of ePT, lPT, and nMLF/aHB populations) and $\mathbf{x}_t \in \mathbb{R}^D$ represents the exogenous covariates. The covariates include variables that drive the system but are not targets of prediction: specifically the instantaneous sensory drive $\Phi(\mathbf{u}_t^{\text{ret}})$ and the behavioral bout state. While the neural populations and bout state correspond to experimentally measurable quantities, the sensory drive is a simulator internal representing processed retinal output. Because the internal bout trigger is stochastic (Eq. S1), the future trajectory contains inherent (aleatoric) uncertainty. Conditioning on $\mathbf{x}$ resolves this, transforming the task from a probabilistic into a deterministic problem.

The goal is to discover a parameterized model f_θ that approximates the mechanisms of f^*. Since the ground-truth mechanisms are unknown in real-world settings, we do not optimize for mechanistic similarity directly. Instead, we employ multi-step forecasting of neural activity as a proxy task. While prediction error is the proxy the search minimizes, our scientific focus is recovering the true mechanisms. The identification task is to find the optimal pair (f, θ) that minimizes the divergence between the recursively simulated trajectory $\hat{\mathbf{a}}$ and the ground truth $\mathbf{a}$ over a horizon $H = 256$:

$$\min_{f,\theta} \sum_{t=1}^{H} \|\hat{\mathbf{a}}_t - \mathbf{a}_t\| \quad \text{s.t.} \quad \hat{\mathbf{a}}_t = f_\theta(\hat{\mathbf{a}}_{t-1}, \mathbf{x}_t). \tag{6}$$

To test for both interpolation and robust extrapolation, we use three of the conditions described in Sec. 3.1 for the in-distribution regime (split chronologically into 70% training, 10% validation, and 20% test), while the fourth is reserved as an out-of-distribution (OOD) holdout to evaluate whether models generalize. The training split is used to fit parameters θ; the validation split scores candidates

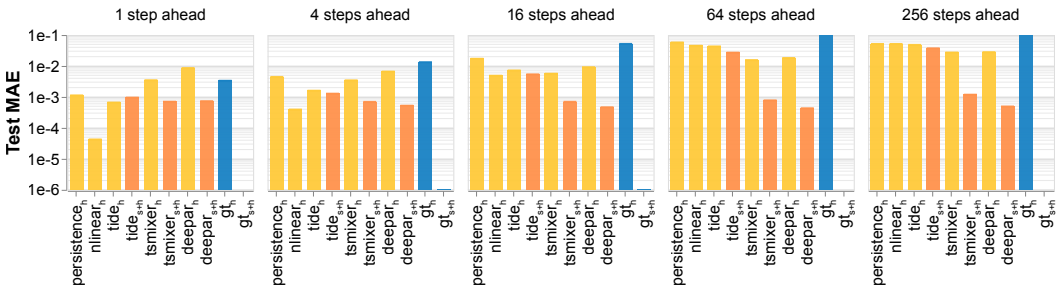

Figure 3: **Sensory information is a prerequisite for system identification.** Performance (Test MAE, log scale) of baseline models and the `gt` circuit across prediction horizons. While models marked with `s+h` are conditioned on exogenous sensory drive, models with `h` only use past history. $\texttt{gt}_\texttt{h}$ yields higher error than the naive $\texttt{persistence}_\texttt{h}$ baseline, suggesting that the true solution is non-identifiable without sensory drive based on standard error metrics.

and therefore drives code changes; test and holdout are never seen by the search and are used only for the reported evaluation.

Evaluation. We evaluate candidate models on two axes: their ability to predict neural activity and their success in recovering ground-truth mechanisms.

Following ZAPBench [4], we quantify trajectory prediction error using the Mean Absolute Error (MAE) averaged across all units and time steps. For a set of test windows $\mathcal{W}$, the error is defined as:

$$\text{MAE}_{\mathcal{W}} = \frac{1}{|\mathcal{W}|} \sum_{w \in \mathcal{W}} \frac{1}{N} \sum_{n=1}^{N} \frac{1}{H} \sum_{h=1}^{H} |\hat{a}_{h,n}^{w} - a_{h,n}^{w}|. \tag{7}$$

We report this metric on two data splits. The Test MAE measures performance on held-out trajectories from the same stimulus conditions used during training. The Holdout MAE measures performance on the OOD stimulus condition. We posit that a Holdout MAE that is high relative to Test MAE can flag models that likely rely on statistical shortcuts.

Since we operate in an in silico setting with known, differentiable ground truth f^*, we can verify the mechanistic fidelity of a discovered model f_θ: We compute the effective connectivity matrix using Jacobian sensitivity analysis and impulse responses (IR) relative to f^*. This allows us to evaluate recovery of static and dynamic mechanisms, respectively. App. C provides details on these analyses including computation of the error metrics $\mathcal{L}_{\text{Jac}}$ and $\mathcal{L}_{\text{IR}}$. We define a model as *mechanistically faithful* if it demonstrates robust OOD generalization and recovers both static and dynamic mechanisms.

Baseline Models. To contextualize the performance of our discovered models, we compare against two distinct categories of baselines. First, to establish the predictive ceiling of the dataset, we employ state-of-the-art time-series forecasting architectures including `nlinear` [32], `tide` [33], and `tsmixer` [34], as well as a naive `persistence` baseline. These models map a history window directly to the future horizon without explicitly modeling step-by-step transition dynamics. While they serve as rigorous controls for predictive accuracy, they do not yield an explicit transition function. Second, we evaluate `deepar`, a DeepAR-style recurrent network [35]. Like our discovered models, `deepar` recursively predicts the next step via a learned transition function. To capture low-dimensional sequential latent dynamics—an approach popularized by methods like LFADS [36]—we also evaluated a `deepar` variant with an LFADS-inspired bottleneck, though this did not yield predictive improvements. Details are in App. D.

Tree Search. To navigate the combinatorial space of potential neural architectures, we employ an automated code generation approach driven by LLMs and Tree Search: We use the method proposed by Aygün et al. [7], which combines a pre-trained LLM with a heuristic search algorithm [Predictor Upper Confidence Bound; 37] to iteratively refine solutions. The search scores solutions using validation set MAE on in-distribution conditions. We utilized Gemini 3 Flash [38] as the backbone LLM. All details are in App. E.

Code and Data. Code, including the JAX-translated model, and data are available at `https://in-silico-sysid.web.app`.

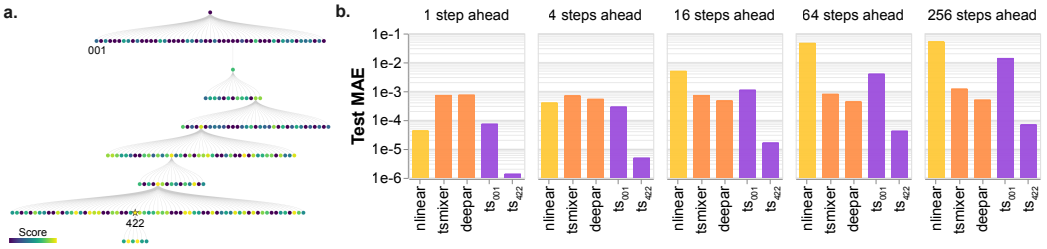

Figure 4: **Tree search discovers SOTA predictive models**. **a.** Progression of LLM-guided tree search, highlighting first ($\texttt{ts}_{001}$) and highest-scoring ($\texttt{ts}_{422}$) solutions, the latter obtained after seven generations of code mutation. For visual clarity, this tree is pruned to show only the ancestral path of $\texttt{ts}_{422}$; the complete, unpruned search tree is in Fig. S1. **b.** Performance comparison of discovered models against human-curated baselines. The highest-scoring discovered tree-search architecture ($\texttt{ts}_{422}$) significantly outperforms baselines across all prediction horizons.

4 Results

4.1 Sensory Drive and Identifiability

A fundamental challenge in neural modeling is distinguishing intrinsic dynamics from responses to extrinsic drive [39]. We used our in silico testbed to perform this decomposition explicitly—an intervention virtually impossible to replicate in real-world experiments. We evaluated a model reflecting the ground truth circuit in two regimes: conditioned on instantaneous sensory drive and the history of past activity ($\texttt{gt}_{\texttt{s+h}}$), and a control for which the sensory drive $\Phi(\mathbf{u}_t^{\text{ret}})$ is set to zero ($\texttt{gt}_{\texttt{h}}$).

Despite using the true transition function f^*, $\texttt{gt}_{\texttt{h}}$ yields higher error than the naive $\texttt{persistence}_{\texttt{h}}$ baseline (which yields an MAE ≈ 0.06 across horizons; Fig. 3). This highlights the dissipative nature of the system: without sensory drive, the underlying circuit dynamics decay to rest. In contrast, the $\texttt{persistence}_{\texttt{h}}$ baseline minimizes error by predicting that neural activity persists unchanged—thereby reflecting the average activity maintained by the continuous sensory drive. This shows that in a purely autoregressive setting, the true mechanistic model is practically non-identifiable based on standard error metrics. Only when conditioning on sensory drive in addition to history is the system identification task well-posed: The error for $\texttt{gt}_{\texttt{s+h}}$ drops to the numerical floor (MAE $\approx 2 \times 10^{-8}$ across horizons). We comment on the implications for real-world benchmarks in Sec. 5.

Access to sensory drive is necessary but not sufficient; the model architecture must also be capable of effectively utilizing it. While the multivariate $\texttt{tsmixer}_{\texttt{s+h}}$ and $\texttt{deepar}_{\texttt{s+h}}$ baselines improve with the addition of sensory covariates across all prediction horizons, the univariate $\texttt{tide}_{\texttt{s+h}}$ model does not. The sensory drive exerts a differential effect across the population. Global univariate models like $\texttt{tide}$—which apply a shared mapping across all units—lack the expressivity to capture how sensory information drives heterogeneous neural responses.

4.2 Model Discovery with Tree Search

The search space of predictive architectures is vast, encompassing a wide variety of structural biases, connectivity patterns, and temporal integration strategies. Traditionally, navigating this space relies on researcher intuition and manual trial-and-error. To evaluate whether automated discovery can identify accurate models of neural dynamics, we utilized an LLM-guided tree search.

In our initial experiments, tree search tended to produce heterogeneous code across various frameworks, making systematic evaluation and integration difficult. We developed a standardized evaluation harness and prompt designed specifically to search within the class of autoregressive models implemented in `flax` [40]. To ensure the discovery process remained computationally efficient, we implemented a heavily optimized, GPU-resident data loading pipeline. We found that training models on a truncated rollout of 32 timesteps was sufficient to identify architectures that performed well on the full 256-step prediction horizon, significantly reducing the training time required for each candidate solution during the search process.

Following these changes, the automated discovery process (Fig. 4a) was highly successful, with many of the identified solutions achieving significantly lower Test MAE than the predictive baselines:

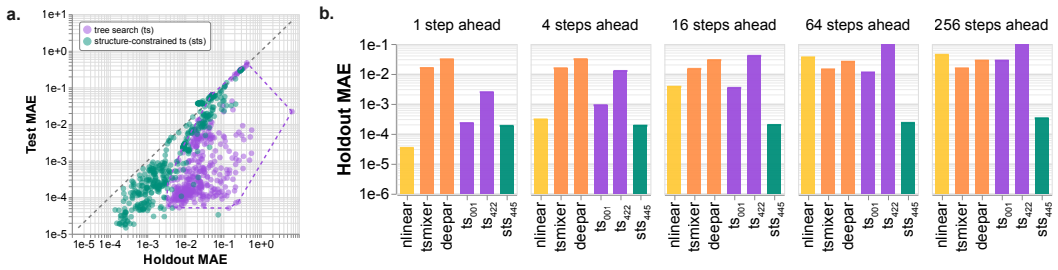

Figure 5: **Structural priors enable robust generalization. a.** Generalization analysis comparing performance on in-distribution (Test MAE) versus out-of-distribution stimuli (Holdout MAE). Tree search models (`ts`, purple) exhibit a significant generalization gap; while they achieve high accuracy on the test set, they fail to generalize to novel stimuli, suggesting they memorize sensory-motor correlations. Structurally-constrained tree search solutions (`sts`, green) cluster closer along the diagonal, indicating robust transfer to unseen environments. Dotted purple line marks the convex hull for `ts` solutions. **b.** Holdout performance across prediction horizons. The highest-scoring constrained model ($\texttt{sts}_{445}$) significantly outperforms its unconstrained counterpart ($\texttt{ts}_{422}$) and baselines.

Fig. 4b highlights the performance of two discovered models, $\texttt{ts}_{001}$ (the first AI-generated solution) and $\texttt{ts}_{422}$, compared to `nlinear`, `tsmixer`, and `deepar`. Notably, $\texttt{ts}_{422}$ represents the highest-scoring outcome of our search, outperforming state-of-the-art baselines by almost an order of magnitude when averaging over all prediction horizons.

4.3 Generalization and Interpretability

Despite the high predictive accuracy of the tree-search models, they have limited scientific utility: they fail to generalize and it is unclear how to map them to the underlying circuit.

As shown in Fig. 5a, unconstrained tree-search models (`ts`) with low Test MAE exhibit a significant generalization gap. Although they achieve near-perfect MAE on the test set for trained stimulus conditions, their performance degrades sharply when evaluated on held-out visual stimuli. This indicates that the search identifies solutions that exploit stimulus-specific correlations—essentially capturing statistical shortcuts—rather than recovering mechanisms that govern the zebrafish's response.

Furthermore, these solutions offer limited interpretability for the computational neuroscientist. In App. F.3, we show the transition function for the highest-scoring model, $\texttt{ts}_{422}$. While the code is mathematically precise, it is mechanistically opaque. The transition function lacks a clear correspondence to the neural circuit and instead makes its predictions in a 256-dimensional latent space. This lack of mechanistic transparency makes it difficult to map the model's internal operations back to the physical circuit, rendering it of limited value for formulating testable biological hypotheses.

4.4 Incorporating Structural Priors

To bridge the gap between predictive accuracy and scientific utility, we modified the discovery process by incorporating structural priors. Instead of an unconstrained search over transition functions, we provided the tree search with prior information derived from the structural connectivity of the neural circuit (App. E.3): These hints reveal the system's hierarchical organization, the sparse connectivity patterns (including recurrence) and the functional role of covariates. Crucially, the structural prior provided to the search contains no information about connectivity weights—only about the existence and absence of connections. The specific non-linear functions and the precise connectivity weights that govern the interactions must still be autonomously discovered.

The impact of the structural prior is clearly evident in the model's ability to generalize. As shown in Fig. 5a, structure-constrained models (green markers, `sts`) are closer to the diagonal region. Unlike the unconstrained models, `sts` solutions maintain more consistent performance across both training and held-out stimulus conditions, suggesting they overfit less to correlations present in train and test sets. The highest-scoring model ($\texttt{sts}_{445}$) is significantly more robust on the OOD condition, more than an order of magnitude for longer prediction horizons compared to baselines (Fig. 5b). App. F.4 shows the transition function for $\texttt{sts}_{445}$: In contrast to the hard-to-interpret code of $\texttt{ts}_{422}$, it reflects the structural organization of the circuit and is composed of semantically meaningful motifs.

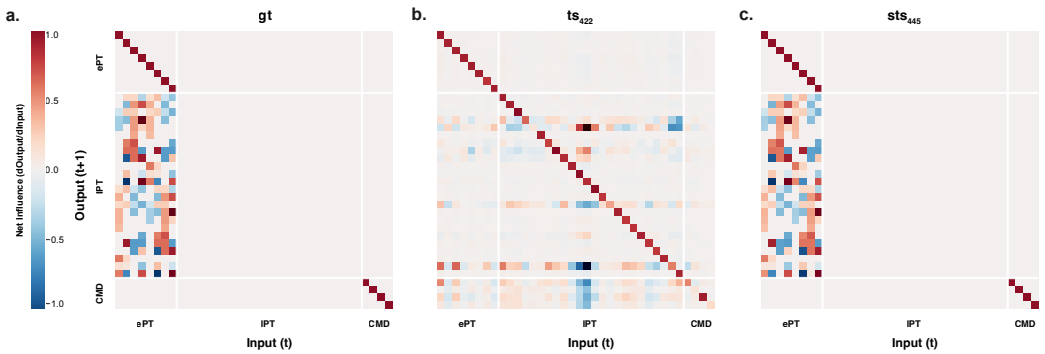

Figure 6: **Structural priors enable mechanistic recovery. a.** Effective connectivity matrices derived via Jacobian sensitivity analysis for the ground-truth circuit. Note the distinct excitatory (red) and inhibitory (blue) pattern in the ePT $\rightarrow$ lPT interactions (middle-left block) and the absence of recurrence in the lPT block (center). **b.** The unconstrained $\texttt{ts}_{422}$ model largely fails to capture the ePT $\rightarrow$ lPT interactions, instead relying on spurious recurrent lPT dynamics. **c.** The structure-constrained model, which received a boolean mask for ePT $\rightarrow$ lPT connectivity but no information about strength, faithfully recovers the sign and magnitude pattern of the interactions.

However, this success depends on the underlying language model; an identical search with a less capable LLM (Gemini 2.5 Flash; [41]) failed to discover generalizing solutions (Fig. S4).

4.5 Mechanistic Recovery

We analyzed the effective connectivity of the highest-scoring discovered solution to evaluate mechanistic recovery (App. C.1). In the ground truth circuit (Fig. 6a), the lPT population depends on structured inputs from the ePT population (middle-left block). The unconstrained $\texttt{ts}_{422}$ model (Fig. 6b) fails to recover these upstream dependencies. Instead, it relies on spurious recurrence within the lPT population (central block)—effectively substituting internal memory for sensory processing.

In contrast, $\texttt{sts}_{445}$ recovers the correct effective connectivity. While the structural prior included a binary matrix for ePT $\rightarrow$ lPT connectivity and indicated absence of recurrent lPT connections, it contained no information about the strengths of the feedforward interactions. Consequently, the optimization process had to discover the nature of these links. We find that $\texttt{sts}_{445}$ correctly recovered the excitatory and inhibitory checkerboard pattern of ePT $\rightarrow$ lPT interactions, matching the ground truth in both sign and relative magnitude (compare Fig. 6a versus Fig. 6c).

To analyze the fidelity of dynamic mechanisms of the discovered models, we performed an impulse response analysis (App. C.2), perturbing the system to a saturated state and observing the relaxation dynamics under different behavioral contexts (swimming versus resting): While $\texttt{ts}_{422}$ produces spurious oscillations and fails to capture the correct decay rates, $\texttt{sts}_{445}$ closely tracks the ground truth trajectories as shown in Fig. S2.

We confirm the qualitative differences discussed in this section by quantitative metrics reported in Tab. S1: We find that, relative to $\texttt{ts}_{422}$, $\texttt{sts}_{445}$ achieves a more than 30-fold reduction in effective connectivity error ($\mathcal{L}_{\text{Jac}}$), and a more than 5-fold reduction in impulse response error ($\mathcal{L}_{\text{IR}}$). Tab. S2 reports the same metrics for the top 50 highest-scoring solutions of both tree searches, confirming the robustness of these results. Finally, we demonstrate that the structure-constrained search is effective even when the provided structural prior is noisy or incomplete (App. E.4). Furthermore, we investigated the impact of observation noise, finding that mechanistic recovery gracefully degrades as the signal-to-noise ratio decreases, with recovery ultimately failing at high noise levels (App. E.5).

Taken together, these analyses show that structural priors enable the discovery of models that abandon statistical shortcuts in favor of recovering true mechanisms, allowing faithful system identification.

5 Discussion

Our analysis reveals that sensory drive is a prerequisite for identifiability, while in-distribution accuracy is a poor proxy for system identification. Crucially, we find that structural priors enable

mechanistic recovery, eliminating statistical shortcuts that unconstrained models exploit. These insights motivate three concrete recommendations for future modeling of real-world datasets:

#1: Redefining prediction tasks. Current benchmarks, such as ZAPBench [4] which is based on whole-brain neural activity recordings from a larval zebrafish, treat neural activity as a self-contained time series suitable for autoregressive prediction. Our control experiments challenge this practice by demonstrating that ablating sensory drive causes non-identifiability of the true circuit mechanisms, rather than mere performance degradation. In real-world recordings, this issue is exacerbated by additional unobserved drivers (e.g., olfactory, auditory, or mechanosensory inputs). We therefore propose redefining the task: rather than autoregressively predicting the entire network's activity from history and visual stimulus, we suggest using known "upstream" sensory neurons—e.g., identified via the connectome—as conditioning inputs, and forecasting the activity of all remaining non-sensory neurons situated downstream in the sensory-to-motor hierarchy. In datasets lacking recordings of sensory neurons, one could trace afferents to their immediate downstream partners and condition on their measured activity for a better-posed system identification task.

#2: Prioritize OOD generalization. We show that unconstrained models can achieve near-perfect accuracy on familiar data by exploiting statistical shortcuts, only to fail completely when the stimulus changes. To ensure models capture true mechanisms rather than dataset-specific correlations, evaluation should prioritize performance on held-out stimulus classes, and perturbation studies.

#3: Use physical wiring as a scaffold. We propose a shift toward structural grey-box modeling. While black-box models currently dominate neural forecasting, our in silico experiments suggest they poorly recover mechanisms and tend to have limited interpretability.

The upcoming release of the synapse-resolution connectome for the ZAPBench specimen will make these recommendations practical and offer an opportunity to test a key result from our work: that static wiring diagrams can serve as a potent scaffold. It is often assumed that recovering mechanisms requires detailed knowledge of synaptic strengths and valences from the ultrastructure or molecular annotations [42, 43]. However, we find that knowing the wiring diagram provides a structural prior that allows an optimization process to infer mechanisms directly from paired neural activity.

Limitations and Future Work. simZFish is a simplification of the vertebrate brain, focusing on a particular visuomotor circuit and associated behavior. Future work could extend the simulator to include broader behavioral repertoires and associated neural circuits. This would allow us to test how well our findings extrapolate to larger-scale systems. We expect our results regarding identifiability and statistical shortcuts to hold, as they are structural properties of the task: unobserved sensory drive fundamentally breaks identifiability regardless of network size, and larger hypothesis spaces likely exacerbate the risk of shortcuts. However, our mechanistic recovery result is plausibly scale-dependent. The degree to which a binary connectivity prior constrains the search depends on the fraction of the hypothesis space it eliminates, which changes with network size N. We do not have evidence that a binary wiring diagram remains a sufficient scaffold for circuits comprising thousands of units, let alone a whole larval zebrafish brain ($\sim$100,000 neurons). Additionally, while the tree search is applicable to high-dimensional data, the process is computationally intensive.

We prioritized a fully observable, high-fidelity regime to rigorously validate mechanistic recovery against a known ground truth. Extending this to settings with partial observability, or broader OOD evaluations across diverse stimuli are areas for future research. We cannot at present state the exact experimental requirements for replacing our privileged sensory covariate with a measured signal.

While we isolated the effect of the prior via the ablation between our tree searches, developing hand-specified grey-box baselines could provide a valuable intermediate comparison. Finally, our automated discovery relied on Gemini 3 Flash. An interesting open question for future work is whether more capable frontier models can identify faithful mechanistic solutions with fewer iterations.

Conclusion. *In silico* testbeds allow us to better understand the capabilities of AI-driven code search. Our results reveal that treating model discovery purely as an unconstrained error-minimization problem encourages solutions that exploit statistical shortcuts rather than faithful system identification. As we deploy these tools on complex real-world science problems, testbeds with verifiable solutions can guide the design of constraints and metrics needed to align AI-driven discovery with scientific utility and mechanistic understanding.

Acknowledgements

We would like to thank Peter H. Li, Franz Rieger, Zhengdao Chen, and the Science AI team at Google Research for discussions and project support. We also thank the authors of open-source software enabling this work, including simZFish [5], Webots [29], JAX [31], Flax [40], and the wider Python community.

References

[1] Martin Schrimpf, Jonas Kubilius, Michael J. Lee, N. Apurva Ratan Murty, Robert Ajemian, and James J. DiCarlo. Integrative Benchmarking to Advance Neurally Mechanistic Models of Human Intelligence. *Neuron*, 2020. URL https://doi.org/10.1016/j.neuron.2020.07.040.

[2] Felix Pei, Joel Ye, David Zoltowski, Anqi Wu, Raeed H. Chowdhury, Hansem Sohn, Joseph E. O'Doherty, Krishna V. Shenoy, Matthew T. Kaufman, Mark Churchland, Mehrdad Jazayeri, Lee E. Miller, Jonathan Pillow, Il Memming Park, Eva L. Dyer, and Chethan Pandarinath. Neural Latents Benchmark '21: Evaluating latent variable models of neural population activity. In *NeurIPS*, 2021. URL https://doi.org/10.48550/arXiv.2109.04463.

[3] Polina Turishcheva, Paul G. Fahey, Laura Hansel, Rachel Froebe, Kayla Ponder, Michaela Vystrčilová, Konstantin F. Willeke, Mohammad Bashiri, Eric Wang, Zhiwei Ding, Andreas S. Tolias, Fabian H. Sinz, and Alexander S. Ecker. The dynamic sensorium competition for predicting large-scale mouse visual cortex activity from videos. *arXiv preprint*, 2023. URL https://doi.org/10.48550/arXiv.2305.19654.

[4] Jan-Matthis Lueckmann, Alexander Immer, Alex Bo-Yuan Chen, Peter H. Li, Mariela D Petkova, Nirmala A Iyer, Luuk Willem Hesselink, Aparna Dev, Gudrun Ihrke, Woohyun Park, Alyson Petruncio, Aubrey Weigel, Wyatt Korff, Florian Engert, Jeff Lichtman, Misha Ahrens, Michal Januszewski, and Viren Jain. ZAPBench: A benchmark for whole-brain activity prediction in zebrafish. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=oCHsDpyawq.

[5] Xiangxiao Liu, Matthew D Loring, Luca Zunino, Kaitlyn E Fouke, François A Longchamp, Alexandre Bernardino, Auke J Ijspeert, and Eva A Naumann. Artificial embodied circuits uncover neural architectures of vertebrate visuomotor behaviors. *Science Robotics*, 2025. URL https://doi.org/10.1126/scirobotics.adv4408.

[6] Lennart Ljung. Perspectives on system identification. *Annual Reviews in Control*, 2010. URL https://doi.org/10.1016/j.arcontrol.2009.12.001.

[7] Eser Aygün, Anastasiya Belyaeva, Gheorghe Comanici, Marc Coram, Hao Cui, Jake Garrison, Renee Johnston, Anton Kast, Cory Y. McLean, Peter Norgaard, Zahra Shamsi, David Smalling, James Thompson, Subhashini Venugopalan, Brian P. Williams, Chujun He, Sarah Martinson, Martyna Plomecka, Lai Wei, Yuchen Zhou, Qian-Ze Zhu, Matthew Abraham, Erica Brand, Anna Bulanova, Jeffrey A. Cardille, Chris Co, Scott Ellsworth, Grace Joseph, Malcolm Kane, Ryan Krueger, Johan Kartiwa, Dan Liebling, Jan-Matthis Lueckmann, Paul Raccuglia, Xuefei Wang, Katherine Chou, James Manyika, Yossi Matias, John C. Platt, Lizzie Dorfman, Shibl Mourad, and Michael P. Brenner. An AI system to help scientists write expert-level empirical software. *Nature*, 2026. URL https://doi.org/10.1038/s41586-026-10658-6.

[8] Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *Science*, 324, 2009. URL https://doi.org/10.1126/science.1165893.

[9] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=r1Ue8Hcxg.

[10] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 2024. URL https://doi.org/10.1038/s41586-023-06924-6.

[11] Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint*, 2025. URL https://doi.org/10.48550/arXiv.2506.13131.

[12] Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. ShinkaEvolve: Towards open-ended and sample-efficient program evolution. *arXiv preprint*, 2025. URL https://doi.org/10.48550/arXiv.2509.19349.

[13] Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. Aide: Ai-driven exploration in the space of code. *arXiv preprint*, 2025. URL https://doi.org/10.48550/arXiv.2502.13138.

[14] Pablo Samuel Castro, Nenad Tomasev, Ankit Anand, Navodita Sharma, Rishika Mohanta, Aparna Dev, Kuba Perlin, Siddhant Jain, Kyle Levin, Noemi Elteto, Will Dabney, Alexander Novikov, Glenn C Turner, Maria K Eckstein, Nathaniel D. Daw, Kevin J Miller, and Kim Stachenfeld. Discovering symbolic cognitive models from human and animal behavior. In *International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=dhRXGWJ027.

[15] Reilly Tilbury, Dabin Kwon, Ali Haydaroglu, Jacob Ratliff, Valentin Schmutz, Matteo Carandini, Kevin Miller, Kim Stachenfeld, and Kenneth D. Harris. Characterizing neuronal population geometry with AI equation discovery. *bioRxiv preprint*, 2025. URL https://doi.org/10.1101/2025.11.12.688086.

[16] Eric Jonas and Konrad Paul Kording. Could a neuroscientist understand a microprocessor? *PLoS computational biology*, 2017. URL https://doi.org/10.1371/journal.pcbi.1005268.

[17] Yena Han, Tomaso A Poggio, and Brian Cheung. System identification of neural systems: If we got it right, would we know? In *International Conference on Machine Learning*, 2023. URL https://openreview.net/forum?id=NkTEhPQCjg.

[18] Gopal P Sarma, Chee Wai Lee, Tom Portegys, Vahid Ghayoomie, Travis Jacobs, Bradly Alicea, Matteo Cantarelli, Michael Currie, Richard C Gerkin, Shane Gingell, Padraig Gleeson, R Gordon, R M Hasani, G Idili, S Khayrulin, D Lung, A Palyanov, M Watts, and S D Larson. OpenWorm: Overview and recent advances in integrative biological simulation of caenorhabditis elegans. *Philosophical Transactions of the Royal Society B*, 2018. URL https://doi.org/10.1098/rstb.2017.0382.

[19] Mengdi Zhao, Ning Wang, Xinrui Jiang, Xiaoyang Ma, Haixin Ma, Gan He, Kai Du, Lei Ma, and Tiejun Huang. An integrative data-driven model simulating c. elegans brain, body and environment interactions. *Nature Computational Science*, 2024. URL https://doi.org/10.1038/s43588-024-00738-w.

[20] Victor Lobato-Rios, Shravan Tata Ramalingasetty, Pembe Gizem Özdil, Jonathan Arreguit, Auke Jan Ijspeert, and Pavan Ramdya. NeuroMechFly, a neuromechanical model of adult drosophila melanogaster. *Nature Methods*, 2022. URL https://doi.org/10.1038/s41592-022-01466-7.

[21] Roman Vaxenburg, Igor Siwanowicz, Josh Merel, Alice A Robie, Carmen Morrow, Guido Novati, Zinovia Stefanidi, Gert-Jan Both, Gwyneth M Card, Michael B Reiser, Matthew M Botvinick, Kristin M Branson, Yuval Tassa, and Srinivas C Turaga. Whole-body physics simulation of fruit fly locomotion. *Nature*, 2025. URL https://doi.org/10.1038/s41586-025-09029-4.

[22] Christos Sourmpis, Carl CH Petersen, Wulfram Gerstner, and Guillaume Bellec. Biologically informed cortical models predict optogenetic perturbations. *eLife*, 2025. URL https://doi.org/10.7554/eLife.106827.2.

[23] Lu Mi, Richard Xu, Sridhama Prakhya, Albert Lin, Nir Shavit, Aravinthan Samuel, and Srinivas C Turaga. Connectome-constrained latent variable model of whole-brain neural activity. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=CJzi3dRlJE-.

[24] Jascha Achterberg, Danyal Akarca, DJ Strouse, John Duncan, and Duncan E Astle. Spatially embedded recurrent neural networks reveal widespread links between structural and functional neuroscience findings. *Nature Machine Intelligence*, 2023. URL https://doi.org/10.1038/s42256-023-00748-9.

[25] Janne K Lappalainen, Fabian D Tschopp, Sridhama Prakhya, Mason McGill, Aljoscha Nern, Kazunori Shinomiya, Shin-ya Takemura, Eyal Gruntman, Jakob H Macke, and Srinivas C Turaga. Connectome-constrained networks predict neural activity across the fly visual system. *Nature*, 2024. URL https://doi.org/10.1038/s41586-024-07939-3.

[26] Manuel Beiran and Ashok Litwin-Kumar. Prediction of neural activity in connectome-constrained recurrent networks. *Nature Neuroscience*, 2025. URL https://doi.org/10.1038/s41593-025-02080-4.

[27] Sunny Duan, Ling Liang Dong, and Ila Fiete. From synapses to dynamics: Obtaining function from structure in a connectome constrained model of the head direction circuit. *bioRxiv preprint*, 2025. URL https://doi.org/10.1101/2025.05.26.655406.

[28] Xiuye Chen, Yu Mu, Yu Hu, Aaron T Kuan, Maxim Nikitchenko, Owen Randlett, Alex B Chen, Jeffery P Gavornik, Haim Sompolinsky, Florian Engert, and Misha B Ahrens. Brain-wide organization of neuronal activity and convergent sensorimotor transformations in larval zebrafish. *Neuron*, 2018. URL https://doi.org/10.1016/j.neuron.2018.09.042.

[29] O. Michel. Webots: Professional mobile robot simulation. *Journal of Advanced Robotics Systems*, 2004. URL https://doi.org/10.5772/5618.

[30] Eva A Naumann, James E Fitzgerald, Timothy W Dunn, Jason Rihel, Haim Sompolinsky, and Florian Engert. From whole-brain data to functional circuit models: The zebrafish optomotor response. *Cell*, 2016. URL https://doi.org/10.1016/j.cell.2016.10.019.

[31] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: Composable transformations of Python+NumPy programs, 2018. URL https://github.com/jax-ml/jax.

[32] Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting? In *AAAI Conference on Artificial Intelligence*, 2023. URL https://doi.org/10.1609/aaai.v37i9.26317.

[33] Abhimanyu Das, Weihao Kong, Andrew Leach, Shaan Mathur, Rajat Sen, and Rose Yu. Long-term forecasting with TiDE. *Transactions on Machine Learning Research*, 2023. URL https://openreview.net/forum?id=pCbC3aQB5W.

[34] Si-An Chen, Chun-Liang Li, Nate Yoder, Sercan O Arik, and Tomas Pfister. TSMixer: An all-MLP architecture for time series forecasting. *Transactions on Machine Learning Research*, 2023. URL https://openreview.net/forum?id=wbpxTuXgm0.

[35] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 2020. URL https://doi.org/10.1016/j.ijforecast.2019.07.001.

[36] Chethan Pandarinath, Daniel J O'Shea, Jace Collins, Rafal Jozefowicz, Sergey D Stavisky, Jonathan C Kao, Eric M Trautmann, Matthew T Kaufman, Stephen I Ryu, Leigh R Hochberg, Jaimie M Henderson, Krishna V Shenoy, L F Abbott, and David Sussillo. Inferring single-trial neural population dynamics using sequential auto-encoders. *Nature methods*, 2018. URL https://doi.org/10.1038/s41592-018-0109-9.

[37] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 2016. URL https://doi.org/10.1038/nature16961.

[38] Google. Gemini 3 Flash: Frontier intelligence built for speed, December 17 2025. URL https://blog.google/products-and-platforms/products/gemini/gemini-3-flash/. The Keyword.

[39] Parsa Vahidi, Omid G. Sani, and Maryam M. Shanechi. Modeling and dissociation of intrinsic and input-driven neural population dynamics underlying behavior. *Proceedings of the National Academy of Sciences*, 2024. URL https://doi.org/10.1073/pnas.2212887121.

[40] Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. Flax: A neural network library and ecosystem for JAX, 2024. URL https://github.com/google/flax.

[41] Google. Developers can now start building with Gemini 2.5 Flash, April 17 2025. URL https://blog.google/products-and-platforms/products/gemini/gemini-2-5-flash-preview/. The Keyword.

[42] Cornelia I Bargmann and Eve Marder. From the connectome to brain function. *Nature Methods*, 2013. URL https://doi.org/10.1038/nmeth.2451.

[43] Nils Eckstein, Alexander Shakeel Bates, Andrew Champion, Michelle Du, Yijie Yin, Philipp Schlegel, Alicia Kun-Yang Lu, Thomson Rymer, Samantha Finley-May, Tyler Paterson, Ruchi Parekh, Sven Dorkenwald, Arie Matsliah, Szi-Chieh Yu, Claire McKellar, Amy Sterling, Katharina Eichler, Marta Costa, Sebastian Seung, Mala Murthy, Volker Hartenstein, Gregory S X E Jefferis, and Jan Funke. Neurotransmitter classification from electron microscopy images at synaptic sites in drosophila melanogaster. *Cell*, 2024. URL https://doi.org/10.1016/j.cell.2024.03.016.

Appendices

A Glossary **16**

B Neuromechanical Simulations **17**

C Mechanistic Verification Metrics **19**

C.1 Effective Connectivity Analysis 19

C.2 Impulse Response Analysis 19

D Baseline Models **20**

D.1 `persistence` 20

D.2 `nlinear` 20

D.3 `tide` 20

D.4 `tsmixer` 20

D.5 `deepar` 20

D.6 LFADS-inspired `deepar` variant 20

D.7 Hyperparameter Optimization & Compute Budget 20

E Tree Search **21**

E.1 Search Algorithm 21

E.2 Implementation Details 21

E.3 Prompts 21

E.4 Robustness to Imperfect Structural Priors 23

E.5 Robustness to Observation Noise 24

F Code Listings **25**

F.1 Initial Code for Tree Search 25

F.2 Solution $\texttt{ts}_{001}$ 25

F.3 Solution $\texttt{ts}_{422}$ 26

F.4 Solution $\texttt{sts}_{445}$ 28

G Supplementary Figures **31**

H Supplementary Tables **35**

I Broader Impact **36**

References **37**

A Glossary

System Identification: The process of recovering the underlying mathematical equations or mechanisms of a dynamical system from observed data [1].

Mechanistic Recovery: Faithfully reconstructing the true internal operations and dynamics of the ground-truth system, rather than merely predicting its outputs.

Mechanistically Faithful: A model is considered mechanistically faithful if it achieves robust out-of-distribution (OOD) generalization and successfully recovers both static mechanisms (effective connectivity) and dynamic mechanisms (impulse responses).

Identifiability: A property of a system where its true underlying parameters or mechanisms can be uniquely determined from the available observed data. If a system is *non-identifiable*, multiple different models can yield the exact same predictive error.

Statistical Shortcuts: When a model achieves high predictive accuracy by exploiting dataset-specific correlations rather than learning the true underlying rules of the system.

Structural Priors: Prior knowledge restricted strictly to the physical wiring diagram. In this work, it includes the group hierarchy, whether a dependency is instantaneous or lagged, and a binary existence mask for connections. Crucially, it strictly excludes connection weights, signs (excitatory vs. inhibitory), non-linearities, or time constants.

Effective Connectivity: A measure of the directional influence of one neuron's state on another's, quantified in this work via Jacobian sensitivity analysis.

B Neuromechanical Simulations

We detail the numerical values for the parameters introduced in the neural circuit model (Sec. 3.1), which are based on the reference implementation [2].

Early Pretectum (ePT). The ePT dynamics described in Eq. 2 are governed by the update rate α_t and the response function parameters $\boldsymbol{\omega}_{\text{ept}}, \mathbf{b}_{\text{ept}}$:

$$\begin{aligned}
\alpha_t &= \begin{cases} 0.002 & \text{if swim bout active,} \\ 0.01 & \text{otherwise,} \end{cases} \\
\boldsymbol{\omega}_{\text{ept}} &= [0.068, 0.100, 0.068, 0.100, 0.068, 0.100, 0.068, 0.100]^\top, \\
\mathbf{b}_{\text{ept}} &= [90.0, 60.0, 90.0, 60.0, 90.0, 60.0, 90.0, 60.0]^\top.
\end{aligned}$$

The ePT state vector is ordered as $\mathbf{h}^{\text{ept}} = [\text{ePT}^{\text{S}}_{\text{L}}, \text{ePT}^{\text{A}}_{\text{L}}, \text{ePT}^{\text{I}}_{\text{L}}, \text{ePT}^{\text{P}}_{\text{L}}, \text{ePT}^{\text{S}}_{\text{R}}, \text{ePT}^{\text{A}}_{\text{R}}, \text{ePT}^{\text{I}}_{\text{R}}, \text{ePT}^{\text{P}}_{\text{R}}]^\top$, i.e., grouped by hemisphere (Left, Right) and functional tuning (Superior, Anterior, Inferior, Posterior).

Late Pretectum (lPT). The lPT activations in Eq. 3 are computed using the connectivity matrix $\mathbf{W}_{\text{lpt}}$, the gating mask $\mathbf{M}_{\text{gate}}$, and response parameters $\boldsymbol{\omega}_{\text{lpt}}, \mathbf{b}_{\text{lpt}} \in \mathbb{R}^{24}$:

$$\mathbf{W}_{\text{lpt}} = \begin{bmatrix}
0 & 1 & 1 & -1.5 & 0.8 & 1.5 & -1 & -2 \\
0 & -1 & 1 & 1.5 & 0.8 & 0 & -1 & 0 \\
-0.8 & 1 & 1 & 0 & 0.8 & 1 & -1 & -2 \\
-0.8 & -1 & 1 & 2.5 & 0.8 & 0 & -1 & 0 \\
-0.8 & 0 & 1 & 0.5 & 0.8 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0.8 & 0 & 0 & 0 \\
0 & 0.8 & 1 & 0 & 0 & 0 & 0 & -0.8 \\
0 & 1 & 1 & -1 & 0 & 1.5 & -1 & -1 \\
0 & -1.5 & 1 & 1.5 & 0 & 0 & -1 & 0.5 \\
0 & 0 & 0 & 0 & 0.8 & 0.3 & 0 & 0 \\
0 & 1.5 & 0 & -1 & 0 & 0.5 & 0 & -2 \\
0 & -1.5 & 0 & 1.5 & 0 & -1 & 0 & 1 \\
0.8 & 1.5 & -1 & -2 & 0 & 1 & 1 & -1.5 \\
0.8 & 0 & -1 & 0 & 0 & -1 & 1 & 1.5 \\
0.8 & 1 & -1 & -2 & -0.8 & 1 & 1 & 0 \\
0.8 & 0 & -1 & 0 & -0.8 & -1 & 1 & 2.5 \\
0.8 & 0 & 0 & 0 & -0.8 & 0 & 1 & 0.5 \\
0.8 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & -0.8 & 0 & 0.8 & 1 & 0 \\
0 & 1.5 & -1 & -1 & 0 & 1 & 1 & -1 \\
0 & 0 & -1 & 0.5 & 0 & -1.5 & 1 & 1.5 \\
0.8 & 0.3 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0.5 & 0 & -2 & 0 & 1.5 & 0 & -1 \\
0 & -1 & 0 & 1 & 0 & -1.5 & 0 & 1.5
\end{bmatrix}, \quad \mathbf{M}_{\text{gate}} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}, \quad \boldsymbol{\omega}_{\text{lpt}} = \begin{bmatrix} 2.5 \\ 2.5 \\ 2.5 \\ 2.5 \\ 2.5 \\ 2.5 \\ 4.0 \\ 4.0 \\ 4.0 \\ 5.0 \\ 4.0 \\ 4.0 \\ 2.5 \\ 2.5 \\ 2.5 \\ 2.5 \\ 2.5 \\ 2.5 \\ 4.0 \\ 4.0 \\ 4.0 \\ 5.0 \\ 4.0 \\ 4.0 \end{bmatrix}, \quad \mathbf{b}_{\text{lpt}} = \begin{bmatrix} 0.8 \\ 0.8 \\ 0.8 \\ 0.8 \\ 0.8 \\ 0.8 \\ 0.6 \\ 0.6 \\ 0.6 \\ 0.4 \\ 0.6 \\ 0.6 \\ 0.8 \\ 0.8 \\ 0.8 \\ 0.8 \\ 0.8 \\ 0.8 \\ 0.6 \\ 0.6 \\ 0.6 \\ 0.4 \\ 0.6 \\ 0.6 \end{bmatrix}.$$

The lPT state vector is ordered as

$$\begin{aligned}
\mathbf{h}^{\text{lpt}} = [&\text{lPT}^{\text{oB}_1}_{\text{L}}, \text{lPT}^{\text{oB}_2}_{\text{L}}, \text{lPT}^{\text{B}_1}_{\text{L}}, \text{lPT}^{\text{B}_2}_{\text{L}}, \text{lPT}^{\text{iB}}_{\text{L}}, \text{lPT}^{\text{ioB}}_{\text{L}}, \\
&\text{lPT}^{\text{iMm}}_{\text{L}}, \text{lPT}^{\text{Mm}_1}_{\text{L}}, \text{lPT}^{\text{Mm}_2}_{\text{L}}, \text{lPT}^{\text{oMm}}_{\text{L}}, \text{lPT}^{\text{S}_1}_{\text{L}}, \text{lPT}^{\text{S}_2}_{\text{L}}, \\
&\text{lPT}^{\text{oB}_1}_{\text{R}}, \ldots, \text{lPT}^{\text{S}_2}_{\text{R}}]^\top,
\end{aligned}$$

grouping first by hemisphere (Left, Right) and then by functional type.

The gating function $\mathbf{g}(\mathbf{h}^{\text{ept}}) \in \mathbb{R}^{24}$ referenced in Eq. 3 implements a "soft" AND-gate using a minimum operation between contralateral Superior and ipsilateral Inferior ePT inputs for each hemisphere:

$$g_i(\mathbf{h}^{\text{ept}}) = \begin{cases} \min(0.8 \cdot \text{ePT}^{\text{S}}_{\text{R}},\ \text{ePT}^{\text{I}}_{\text{L}}) & \text{for } i \in \{1, \ldots, 12\} \text{ (left hemisphere),} \\ \min(0.8 \cdot \text{ePT}^{\text{S}}_{\text{L}},\ \text{ePT}^{\text{I}}_{\text{R}}) & \text{for } i \in \{13, \ldots, 24\} \text{ (right hemisphere).} \end{cases}$$

Restricted via $\mathbf{M}_{\text{gate}}$ to the four S-type lPT neurons ($i \in \{11, 12, 23, 24\}$), this term replaces the corresponding linear ePT inputs (whose entries are therefore 0 in $\mathbf{W}_{\text{lpt}}$ above, while the anatomical dependencies remain `True` in the binary `CONNECTIVITY` matrix in App. E.3). This ensures S-type neurons are driven strongly only when both directional conditions are met, scaled by a factor of 0.8 on the contralateral input to account for asymmetric drive.

Command Neurons (nMLF/aHB). The dynamics of the command neurons in Eq. 4 are defined by the update rate $\beta = 0.0005$, connectivity $\mathbf{W}_{\text{cmd}}$, and response parameters $\boldsymbol{\omega}_{\text{cmd}}, \mathbf{b}_{\text{cmd}}$:

$$\mathbf{W}_{\text{cmd}}^{1\ldots12} = \begin{bmatrix} 0.1 & -0.1 & 0.32 & -0.08 & 0.25 & -0.05 & 0.6 & 0.4 & 0 & 0 & 0.3 & -0.1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.8 & 0 & 0 \\ 0.1875 & 0.3875 & 0.2165 & 0.3165 & 0 & 0 & 0 & 0.0165 & 0.4165 & 0 & 0.096 & 0.096 \\ -0.1665 & -0.1665 & -0.1665 & -0.1665 & 0 & 0 & 0 & -0.105 & -0.105 & 0 & -0.29 & -0.29 \end{bmatrix},$$

$$\mathbf{W}_{\text{cmd}}^{13\ldots24} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.8 & 0 & 0 \\ 0.1 & -0.1 & 0.32 & -0.08 & 0.25 & -0.05 & 0.6 & 0.4 & 0 & 0 & 0.3 & -0.1 \\ -0.1665 & -0.1665 & -0.1665 & -0.1665 & 0 & 0 & 0 & -0.105 & -0.105 & 0 & -0.29 & -0.29 \\ 0.1875 & 0.3875 & 0.2165 & 0.3165 & 0 & 0 & 0 & 0.0165 & 0.4165 & 0 & 0.096 & 0.096 \end{bmatrix},$$

$$\boldsymbol{\omega}_{\text{cmd}} = [6.5, 6.5, 4.0, 4.0]^\top, \quad \mathbf{b}_{\text{cmd}} = [0.4, 0.4, 0.65, 0.65]^\top.$$

The resulting command state vector is ordered as $\mathbf{h}_t^{\text{cmd}} = [\text{nMLF}_\text{L}, \text{nMLF}_\text{R}, \text{aHB}_\text{L}, \text{aHB}_\text{R}]^\top$.

Motor Circuits. Locomotion in zebrafish is discrete (burst-and-glide). The "Bout Gate" is implemented as a stochastic integrate-and-fire mechanism. During the inter-bout glide phase (once the active-bout countdown finishes), an evidence accumulator I_t integrates the total drive from the nMLF with rectified Gaussian noise:

$$I_t = I_{t-1} + \gamma \sum_{i=1}^{2} h_{t,i}^{\text{cmd}} + |\epsilon_t|, \quad \epsilon_t \sim \mathcal{N}(0, \sigma^2). \tag{S1}$$

A swim bout is triggered if I_t exceeds a threshold θ_{bout}, resetting the integrator to zero. Upon initiation, the discrete action $c_t \in \{\text{Forward}, \text{Left}, \text{Right}\}$ is sampled from a categorical distribution determined by the command neuron states, with action probabilities $\mathbf{p}_t$:

$$\mathbf{p}_t = \frac{\mathbf{s}_t}{\|\mathbf{s}_t\|_1} \quad \text{with} \quad \mathbf{s}_t = \mathbf{W}_{\text{act}} \mathbf{h}_t^{\text{cmd}} + \mathbf{b}_{\text{act}}. \tag{S2}$$

When the "Behavior Determinator" initiates a turn, it triggers greater activity in the ventral spinal neurons (vSPN) on one side, which in turn leads to increased ipsilateral motor neuron activation and the necessary body curvature for the turn. Finally, twelve spinal central pattern generators (CPGs), modeled as coupled oscillators, generate rhythmic tail undulations, and twelve motor neurons determine body curvature by integrating outputs from vSPNs and CPGs.

The evidence accumulation dynamics in Eq. S1 are defined by the integrator threshold $\theta_{\text{bout}} = 436.0$, noise scale $\sigma = 0.65$, and nMLF input scaling factor $\gamma = 0.6$.

The action probabilities defined in Eq. S2 are computed using the command state vector $\mathbf{h}_t^{\text{cmd}}$, the projection matrix $\mathbf{W}_{\text{act}}$, and $\mathbf{b}_{\text{act}}$:

$$\mathbf{W}_{\text{act}} = \begin{bmatrix} 1.25 & 1.25 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{b}_{\text{act}} = \begin{bmatrix} 0.200 \\ 0.066 \\ 0.066 \end{bmatrix}.$$

For an explanation of all other stages in the model, see Liu et al. [2].

C Mechanistic Verification Metrics

C.1 Effective Connectivity Analysis

To evaluate whether the discovered models recover the true static mechanisms of the neural circuit, we computed the *Effective Connectivity Matrix*. This matrix quantifies the sensitivity of the predicted neural state at time $t+1$ to perturbations in the state at time t (Fig. 6).

Formally, we analyze the one-step transition $\mathbf{a}_{t+1} = f_\theta(\mathbf{a}_t, \mathbf{x}_{t+1})$. The effective connectivity matrix $J \in \mathbb{R}^{N \times N}$ is defined as the Jacobian of the future state with respect to the current state:

$$J_{ij} = \frac{\partial(\mathbf{a}_{t+1})_i}{\partial(\mathbf{a}_t)_j} \tag{S3}$$

where J_{ij} represents the causal influence of source neuron j on target neuron i.

To isolate the intrinsic connectivity of the circuit, we compute J while setting exogenous covariates to zero ($\mathbf{x}_{t+1} = \mathbf{0}$). To account for non-linearities (e.g., sigmoids) where gradients vanish in saturated regions, we do not evaluate J at a single operating point. Instead, we estimate the expected sensitivity by averaging over $K = 11$ homogeneous probe states. We define a set of scalars $\mathcal{S} = \{0.0, 0.1, \ldots, 1.0\}$ and construct input vectors $\mathbf{a}^{(s)} = s \cdot \mathbf{1}$, where $\mathbf{1}$ is the all-ones vector. This artificial grid decouples the evaluation from the training dataset to characterize the interaction structure across the entire continuous dynamic range. The reported connectivity is:

$$\bar{J} = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} J|_{\mathbf{a}_t = \mathbf{a}^{(s)}} \tag{S4}$$

We quantify the mechanistic fidelity of a model by comparing its connectivity $\bar{J}_{\text{model}}$ to the ground truth $\bar{J}_{\text{gt}}$ using the Frobenius norm:

$$\mathcal{L}_{\text{Jac}} = ||\bar{J}_{\text{model}} - \bar{J}_{\text{gt}}||_F \tag{S5}$$

C.2 Impulse Response Analysis

To assess recovery of dynamic mechanisms, we performed an *Impulse Response Analysis*. We probe how the system relaxes from perturbed states back to rest, and how this dynamic is modulated by behavioral states.

We simulate the system for a horizon of $H = 256$ steps starting from a set of impulse conditions. We test $N + 1$ initial conditions: 36 cases where a single neuron is activated ($\mathbf{a}_0 = \mathbf{e}_i$) and one case where the entire system is saturated ($\mathbf{a}_0 = \mathbf{1}$). We evolve these states using two fixed covariates $\mathbf{x}_{\text{mode}}$ to test mechanisms depending on swim state (e.g., the ePT update in Eq. 2):

1. **Rest:** Bout state and sensory drive set to 0.
2. **Swim:** Bout state set to 1, sensory drive set to 0.

We quantify the divergence between the model's predicted response traces ($\hat{\mathbf{A}}$) and the ground truth traces ($\mathbf{A}^*$) using the Mean Absolute Error (MAE) averaged over all time steps, units, and initial conditions. We report this metric separately for the two regimes:

$$\mathcal{L}_{\text{IR}}^{\text{rest}} = \text{MAE}(\hat{\mathbf{A}}^{\text{rest}}, \mathbf{A}^{*\text{rest}}), \quad \mathcal{L}_{\text{IR}}^{\text{swim}} = \text{MAE}(\hat{\mathbf{A}}^{\text{swim}}, \mathbf{A}^{*\text{swim}}) \tag{S6}$$

D Baseline Models

D.1 `persistence`

The naive `persistence` baseline repeats the per-neuron activity from the last timestep across the entire prediction horizon.

D.2 `nlinear`

For `nlinear` [3], we use a model with context window size of 2. For optimization, we use AdamW [4] with a learning rate of 10^{-4} and weight decay of 10^{-4}.

D.3 `tide`

For `tide` [5], we use a model with a context window size of 2, 2 encoder and decoder layers each, a hidden layer size of 128, a decoder output dimensionality of 32, and no instance normalization. For optimization, we use AdamW [4] with a learning rate of 10^{-3} and weight decay of 10^{-4}.

D.4 `tsmixer`

For `tsmixer` [6], we use a model with a context window size of 2, 2 blocks, and an MLP dimension of 256, and no instance norm. For optimization, we use AdamW [4] with a learning rate of 10^{-3} and weight decay of 10^{-4}.

D.5 `deepar`

For `deepar` [7], we use a deterministic point-forecasting adaptation with 2 stacked GRU layers, with a hidden size of 64. For optimization, we use AdamW [4] with a learning rate of 10^{-3} and weight decay of 10^{-4}.

D.6 LFADS-inspired `deepar` variant

To capture low-dimensional sequential latent dynamics, we evaluated a `deepar` variant with a bottleneck, inspired by LFADS [8]. This variant did not improve performance (e.g., with a 16-factor bottleneck, Test MAE was $1.8\times$ higher and Holdout MAE was not improved, at $1.3\times$ that of the unbottlenecked model), so we report the unbottlenecked `deepar` version in the main results. Note that this variant is not equivalent to LFADS, as it has no variational sequential prior and no inferred controller inputs.

D.7 Hyperparameter Optimization & Compute Budget

All baseline models were tuned with grid searches consuming approximately 900 GPU-hours on NVIDIA V100 GPUs. For `nlinear`, `tide`, and `tsmixer`, we swept context window sizes and learning rates, keeping other hyperparameters as defined in ZAPBench [9]. For `deepar`, we swept a broader grid that included layer counts, hidden units, learning rates, and an optional LFADS-inspired bottleneck of varying size.

E Tree Search

We formulate the model discovery process as a search problem where the state space consists of executable Python code defining neural transition functions, and the objective is to maximize predictive accuracy on the validation set.

E.1 Search Algorithm

We utilize the Predictor Upper Confidence Bound applied to Trees (PUCT) algorithm [10, 11] to balance exploration of novel solutions and exploitation (refining high-performing solutions). Unlike standard Monte Carlo Tree Search, where the algorithm traverses from the root to a leaf node for expansion, the space of possible code edits here is effectively infinite. Consequently, the algorithm treats every existing node in the tree as a candidate for expansion.

At each step, a node u^* is sampled from the entire tree $\mathcal{T}$ by maximizing the PUCT objective:

$$u^* = \underset{u \in \mathcal{T}}{\operatorname{argmax}} \left(\tilde{R}(u) + c_{\text{puct}} |\mathcal{T}|^{-1} \frac{\sqrt{\sum_{v \in \mathcal{T}} V(v)}}{1 + V(u)} \right), \tag{S7}$$

where $V(u)$ is the visit count of node u, c_{puct} is an exploration constant, and $|\mathcal{T}|^{-1}$ acts as a flat prior.

The term $\tilde{R}(u)$ represents the normalized rank score of a solution. Solutions are scored by calculating the negative Mean Absolute Error (MAE) on the validation set, averaged over in-distribution conditions. These raw scores are converted into ranks $\text{Rank}_{\mathcal{T}}(u)$ in ascending order (where the best solution with the least error has the highest rank), and normalized as:

$$\tilde{R}(u) = \frac{\text{Rank}_{\mathcal{T}}(u) - 1}{|\mathcal{T}| - 1}. \tag{S8}$$

Upon selecting a node u^*, we prompt the LLM to generate a child node by modifying the source code of u^*. The LLM acts as a mutation operator, guided by instructions defined in App. E.3, and the initial code specified in App. F.1. The generated code is immediately executed in a sandboxed environment to compute the validation score on the task defined in Sec. 3.2. Finally, the visit count V is incremented for the selected node and backpropagated to its ancestors.

E.2 Implementation Details

We utilized Gemini 3 Flash [12] as the backbone LLM for code generation and mutation (see Fig. S4 for an ablation using Gemini 2.5 Flash [13]). Each tree search was limited to a maximum of 500 nodes. To ensure computational feasibility on a single NVIDIA T4 GPU, we enforced a runtime limit of 12 hours per candidate solution. To further accelerate the discovery process, we employed a truncated training regime: models were trained on a short rollout of 32 timesteps. We found this efficient training proxy sufficient to identify architectures that generalize well when rolled out to the full prediction horizon (256 timesteps) used for scoring and evaluation.

LLM costs. As a rough estimate, each tree search consumed less than 5M input/1M output tokens (total over 500 solutions). For this estimate we assume 1 token $\approx$ 4 characters. The unconstrained tree search used 17.6M input/2.9M output characters, whereas the structure-constrained tree search used 13.2M input/3.7M output characters.

Tree-search compute cost. Each tree search was stopped after 500 solutions, with each candidate allocated a Nvidia T4 and a 12-hour limit. The 12-hour limit was rarely reached; median training time was 6h (unconstrained) and 5h (constrained), with $\approx$ 3,200 and $\approx$ 2,700 T4-hours total per search.

E.3 Prompts

The initial prompt for tree search, without prior information about structure:

```
# Predicting neural activity

## Challenge

Your task is to build a forecasting model that **predicts future neural activity traces given past
    activity and covariates**.

**Data Specifications:**
- **Inputs & Outputs:** All neural activity and continuous covariates are normalized to the `[0, 1]`
    range.
- **Covariates (`cov_t`):** Shape `(Batch, CovFeatures=9)`.
  - `cov_t[..., 0]`: **Bout State**. A binary flag (0 or 1). This acts as a switch for the system's
      dynamics.
  - `cov_t[..., 1:]`: **Visual Input**. Continuous signals `[0, 1]` driving the system.

**Key Difficulties:**
1. **Autoregressive Stability:** You will develop a transition model $f(y_{t-1}, c_t) \rightarrow y_t$.
    This is rolled out for 32 timesteps. Small errors at step 1 can explode by step 32.
2. **OOD Generalization:** The test set contains visual patterns **not seen during training**. Your
    model must capture the underlying causal rules, not memorize training sequences.

## Evaluation

Performance is measured by Mean Absolute Error (MAE) on held-out trajectories. Lower is better.

## Rules

1. **Structure:** Keep the sections `CONSTANTS`, `MODEL`, and `OPTIMIZER` in this order.
2. **Interface - Hyperparameters:** You **must** define the following constants in the `CONSTANTS`
    section to control the training harness:
   * `BATCH_SIZE`: (int) Default ~512.
   * `MAX_STEPS`: (int) Default ~500,000.
   * `SCHEDULED_SAMPLING_START_PROB` & `SCHEDULED_SAMPLING_FINAL_PROB`: (float) Controls teacher
       forcing. High start (1.0) helps convergence; low final (<0.1) enforces stability.
   * `SCHEDULED_SAMPLING_DECAY_STEPS`: (int) Steps to decay the probability.
   * `EARLY_STOP_PATIENCE`: (int) Steps to wait before stopping.
   * `MIN_DELTA`: (float) Improvement threshold for early stopping.
3. **Interface - Model:**
   - Use `flax.linen`. Class name must be `Candidate`.
   - `__call__` signature: `(self, past_pred, cov_t, train=False)`.
   - **Shape Contract:** Input `past_pred` is (B, 36). Output must be (B, 36).
4. **Interface - Optimizer:**
   - Use `optax`. Instance must be named `optimizer`.
5. **Constraints:** Do not modify code in `NOTE` comments.

## Strategic Approaches & Hints

To win this competition, you should explore distinct architectural ideas. Do not limit yourself to
    simple MLPs.
```

For the tree search with structural prior, the **Strategic Approaches & Hints** section in the prompt was extended as follows:

```
We have prior knowledge about the neuron hierarchy that is likely critical for a low score. The 36
    neurons are split into three groups with specific dependencies:
1. **EPT Neurons (Indices 0-8):** Depend on their past state (`past_pred[..., 0:8]`) and current
    covariates (`cov_t`). The bout state `cov_t[..., 0]` acts as a switch, modulating the neurons'
    update dynamics based on whether a bout is active. Each neuron i in this group (indices 0-7) is
    driven by a specific visual input: `cov_t[..., 1+i]` (e.g., neuron 0 depends on `cov_t[..., 1]`,
    neuron 1 on `cov_t[..., 2]`, and so on).
2. **LPT Neurons (Indices 8:32):** Depend only on the **newly computed EPT activations** (current step's
    neurons 0-7), not on their own past activations from past_pred. The `CONNECTIVITY` variable below
    indicates sparse connections: `CONNECTIVITY[i][j]` is true if the i-th neuron within the LPT group
    depends on the j-th neuron within the EPT group. Since i ranges from 0-23 and j from 0-7,
    `CONNECTIVITY[0][j]` refers to dependencies of neuron 8 (first LPT neuron), `CONNECTIVITY[1][j]`
    refers to dependencies of neuron 9 (second LPT neuron), and so on.
3. **aHB/nMLF Neurons (Indices 32:36):** Depend on their past state (`past_pred[..., 32:36]`) and the
    **newly computed** LPT activations (current step).
```

... and the following connectivity matrix was included:

```
# NOTE: Connectivity from EPT (cols) to LPT (rows)
CONNECTIVITY = [
 [False, True, True, True, True, True, True],
 [False, True, True, True, True, False, True, False],
 [True, True, True, False, True, True, True, True],
 [True, True, True, True, True, False, True, False],
 [True, False, True, True, True, False, False, False],
 [False, False, True, False, True, False, False, False],
 [False, True, True, False, False, False, False, True],
 [False, True, True, True, False, True, True, True],
 [False, True, True, True, False, False, True, True],
 [False, False, False, False, True, True, False, False],
 [False, True, True, True, True, True, False, True],
 [False, True, True, True, True, True, False, True],
 [True, True, True, True, False, True, True, True],
 [True, False, True, False, False, True, True, True],
 [True, True, True, True, True, True, True, False],
 [True, False, True, False, True, True, True, True],
 [True, False, False, False, True, False, True, True],
 [True, False, False, False, False, False, True, False],
 [False, False, False, True, False, True, True, False],
 [False, True, True, True, False, True, True, True],
 [False, False, True, True, False, True, True, True],
 [True, True, False, False, False, False, False, False],
 [True, True, False, True, False, True, True, True],
 [True, True, False, True, False, True, True, True]]
```

The prompt contains the instructions detailed above and the source code of the selected parent node. It does not include sibling scores, or reasoning traces from other nodes.

E.4 Robustness to Imperfect Structural Priors

Real-world connectomic reconstructions are inherently imperfect. To evaluate the robustness of our approach, we performed additional tree searches with imperfect structural priors. We chose baseline noise levels to mimic conditions seen in dense whole-brain connectomic reconstructions (e.g., a synapse classifier accuracy of $\sim$90% where missing synapses are much more likely than hallucinated connections, or completely missing parts of the matrix corresponding to unimaged or damaged brain regions). These experiments provide directional information on whether and to what degree mechanistic recovery is sensitive to these details.

Specifically, we ran two additional tree searches simulating biological reconstruction errors. In both cases, we supplied the model with a corrupted `CONNECTIVITY` variable in the prompt and appended a warning explicitly informing it about the nature of the noise:

Noisy connectivity ($\texttt{sts}_{\texttt{noisy}}$): We randomly flipped $\sim$10% of the `True` entries in the binary connectivity matrix to `False`, simulating false negatives in synapse detection. We provided this corrupted `CONNECTIVITY` matrix to the model and appended the following warning to the prompt:

```
WARNING: The 'CONNECTIVITY' matrix represents a noisy biological reconstruction. While a 'True'
      indicates a confirmed connection, approximately 10% of the actual connections are missing (false
      negatives). Therefore, 'False' is a strong prior but not an absolute guarantee of no connection.
```

Incomplete connectivity ($\texttt{sts}_{\texttt{incomplete}}$): We entirely removed $\sim$10% of the rows from the connectivity matrix (setting all their entries to `False`) to simulate a structurally incomplete reconstruction. We provided this degraded matrix and added the following warning:

```
WARNING: The connectome reconstruction is structurally incomplete. You will notice that two rows (~10%)
      of the 'CONNECTIVITY' matrix contain only 'False' values. For these specific rows, you cannot rely
      on the matrix.
```

While the search becomes slightly more difficult with these imperfect priors, the scatter plots of solutions in Fig. S3 and the population statistics for the top 50 models (Tab. S3) show that the structure-constrained search remains robust.

E.5 Robustness to Observation Noise

Real-world neural recordings are inherently noisy. To characterize how mechanistic recovery depends on the signal-to-noise ratio (SNR), we generated corrupted copies of the in silico dataset by injecting observation noise into the neural activity traces.

To model realistic temporal correlation, we used an AR(1) Gaussian noise process with a correlation time constant of $\tau = 20$ timesteps. To ensure noise levels were faithful regardless of cross-condition variability in neural dynamics, the noise amplitude was scaled per-unit and per-condition: we computed the empirical standard deviation (σ_{ref}) for each unit within the clean training split of each condition. We evaluated four noise amplitude levels corresponding to 2%, 5%, 10%, and 20% of σ_{ref}.

For these searches, we used the same structure-constrained prompt (App. E.3), with one additional hint appended to the **Strategic Approaches & Hints** section to inform the model about the data degradation:

```
4. **Observation Noise:** Traces of EPT, LPT, and aHB/nMLF neurons have observation noise.
```

Tab. S4 reports the mechanistic recovery metrics for the top 50 highest-scoring models at each noise level. We observe a graceful degradation in performance as the SNR decreases.

F Code Listings

F.1 Initial Code for Tree Search

The initial code, which the tree search was tasked to modify:

```python
#
# CONSTANTS
#

BATCH_SIZE = 512
MAX_STEPS = 500_000
SCHEDULED_SAMPLING_START_PROB = 1.0
SCHEDULED_SAMPLING_FINAL_PROB = 0.01
SCHEDULED_SAMPLING_DECAY_STEPS = 100_000
EARLY_STOP_PATIENCE = 20
MIN_DELTA = 1e-6
LEARNING_RATE = 1e-3


#
# MODEL
#

class Candidate(nn.Module):
  """Candidate transition model for neural activity."""

  @nn.compact
  def __call__(self, past_pred, cov_t, train=False):
    """
    Args:
      past_pred: (Batch, 36) - Previous neural state [0, 1].
      cov_t: (Batch, 9) - Index 0 is Bout {0,1}, 1-8 are Visual [0,1].
      train: bool - Training mode flag.
    Returns:
      (Batch, 36) - Predicted next state.
    """
    bout_state = cov_t[:, 0:1] # Shape (B, 1)
    visual_in = cov_t[:, 1:] # Shape (B, 8)

    # TODO: Implement your architecture here.

    # Identity baseline: predicts no change.
    return past_pred


transition_model = Candidate() # NOTE: Do keep this line as-is.


#
# OPTIMIZER
#

optimizer = optax.adamw(learning_rate=LEARNING_RATE, weight_decay=1e-4)
```

Note that `SCHEDULED_SAMPLING_*_PROB` sets the probability of feeding ground-truth activity rather than the model's own prediction at each rollout step during training, decaying from `START` to `FINAL` over `DECAY_STEPS`. A probability of 1.0 is full teacher forcing; decaying toward 0 trains the model on its own predictions, which can help keep rollouts stable.

F.2 Solution $\texttt{ts}_{001}$

This solution predicts the delta for the next timestep using a residual neural network.

```python
import jax.numpy as jnp
import flax.linen as nn
import optax

#
```

```python
# CONSTANTS
#

BATCH_SIZE = 512
MAX_STEPS = 500_000
# High start prob uses ground truth; low final prob enforces autoregressive stability.
SCHEDULED_SAMPLING_START_PROB = 1.0
SCHEDULED_SAMPLING_FINAL_PROB = 0.02
SCHEDULED_SAMPLING_DECAY_STEPS = 150_000
EARLY_STOP_PATIENCE = 40
MIN_DELTA = 1e-6
LEARNING_RATE = 1e-3


#
# MODEL
#

class Candidate(nn.Module):
  """Residual transition model for neural activity."""

  @nn.compact
  def __call__(self, past_pred, cov_t, train=False):
    """
    Args:
      past_pred: (Batch, 36) - Previous neural state.
      cov_t: (Batch, 9) - Bout switch and Visual inputs.
      train: bool - Training mode.
    Returns:
      (Batch, 36) - Predicted next state.
    """
    # Combine previous state and external drivers
    # Shape: (Batch, 36 + 9) = (Batch, 45)
    x = jnp.concatenate([past_pred, cov_t], axis=-1)

    # MLP for state transition dynamics
    h = nn.Dense(512)(x)
    h = nn.LayerNorm()(h)
    h = nn.silu(h)

    h = nn.Dense(512)(h)
    h = nn.LayerNorm()(h)
    h = nn.silu(h)

    # Predict the delta (residual). Initialize with zeros so
    # the model starts as an identity function baseline.
    delta = nn.Dense(36, kernel_init=nn.initializers.zeros)(h)

    # Autoregressive update: y_t = y_{t-1} + delta
    next_state = past_pred + delta

    # Ensure valid range [0, 1]
    return jnp.clip(next_state, 0.0, 1.0)


transition_model = Candidate() # NOTE: Do keep this line as-is.


#
# OPTIMIZER
#

# Learning rate schedule with warmup and cosine decay for stability
lr_schedule = optax.warmup_cosine_decay_schedule(
    init_value=1e-6,
    peak_value=LEARNING_RATE,
    warmup_steps=2000,
    decay_steps=MAX_STEPS,
)

optimizer = optax.adamw(learning_rate=lr_schedule, weight_decay=1e-4)
```

F.3 Solution $\texttt{ts}_{422}$

This solution predicts in a 256-dimensional latent space: it projects the full state, applies three residual blocks with FiLM modulation from a visual-plus-bout context, and updates via a convex combination

of the previous state and a learned target, plus a small additive delta. No component maps onto a circuit element.

```python
import jax
import jax.numpy as jnp
import flax.linen as nn
import optax

#
# CONSTANTS
#

BATCH_SIZE = 512
MAX_STEPS = 500_000
# Scheduled sampling enforces autoregressive stability.
# We decay slightly slower to allow the model to learn the base rules first.
SCHEDULED_SAMPLING_START_PROB = 1.0
SCHEDULED_SAMPLING_FINAL_PROB = 0.05
SCHEDULED_SAMPLING_DECAY_STEPS = 300_000
EARLY_STOP_PATIENCE = 1000
MIN_DELTA = 1e-9
LEARNING_RATE = 8e-4

#
# MODEL
#

class ResidualBlock(nn.Module):
    features: int

    @nn.compact
    def __call__(self, x, context):
        # Pre-norm architecture for stability
        norm = nn.LayerNorm()(x)

        # Gated Linear Unit (GEGLU variant)
        # Use context to modulate the hidden representation
        h = nn.Dense(self.features * 2, kernel_init=nn.initializers.orthogonal())(norm)
        h_val, h_gate = jnp.split(h, 2, axis=-1)
        h = h_val * nn.gelu(h_gate)

        # FiLM modulation using context
        gamma = nn.Dense(self.features, kernel_init=nn.initializers.zeros)(context)
        beta = nn.Dense(self.features, kernel_init=nn.initializers.zeros)(context)
        h = h * (1.0 + gamma) + beta

        h = nn.Dense(self.features, kernel_init=nn.initializers.orthogonal())(h)
        return x + h

class Candidate(nn.Module):
    """
    Refined Gated Dynamics Model.
    Models neural transitions using learned time-constants and target states,
    heavily conditioned on the behavioral 'Bout' switch.
    """

    @nn.compact
    def __call__(self, past_pred, cov_t, train=False):
        B, N = past_pred.shape # N=36

        # 1. Input Processing
        # Split covariates: Bout State (switch) and Visual Input (driving signal)
        bout = cov_t[:, 0:1]
        visual = cov_t[:, 1:]

        # High-dimensional visual embedding
        v_enc = nn.Dense(128)(visual)
        v_enc = nn.gelu(v_enc)

        # Context incorporates the Bout switch explicitly
        # We use the bout flag to gate visual information
        ctx = jnp.concatenate([
            v_enc,
            bout,
            v_enc * bout, # Bout-specific visual processing
            v_enc * (1.0 - bout) # Non-bout specific visual processing
```

```
        ], axis=-1)
        ctx = nn.Dense(128)(ctx)
        ctx = nn.gelu(ctx)

        # 2. Latent Projection
        # Project current neural state to a latent space
        latent_dim = 256
        x = nn.Dense(latent_dim, kernel_init=nn.initializers.orthogonal())(past_pred)

        # Apply N residual blocks with conditional modulation
        for i in range(3):
            x = ResidualBlock(features=latent_dim, name=f"res_block_{i}")(x, ctx)

        # 3. Liquid-Style Transition
        # We predict a 'gate' (alpha) and a 'target' state.
        # This ensures the output is always a convex combination of previous state
        # and a target, guaranteeing stability in [0, 1].

        # Gate: How much the state changes (1/tau)
        # We initialize it to be small to favor stability early on
        gate_logit = nn.Dense(N, kernel_init=nn.initializers.zeros)(x)
        alpha = nn.sigmoid(gate_logit - 3.0)

        # Target: What the neurons are striving towards
        target = nn.Dense(N, kernel_init=nn.initializers.orthogonal())(x)
        target = nn.sigmoid(target) # Ensure target is in [0, 1]

        # Direct Delta: Small additive term for fast transient dynamics
        # This helps capture sudden shifts that a convex combination might miss.
        delta = nn.Dense(N, kernel_init=nn.initializers.zeros)(x)
        delta = jnp.tanh(delta) * 0.05

        # Update rule: y_t = (1-alpha)*y_{t-1} + alpha*target + delta
        next_state = (1.0 - alpha) * past_pred + alpha * target + delta

        return jnp.clip(next_state, 0.0, 1.0)

transition_model = Candidate() # NOTE: Do keep this line as-is.

#
# OPTIMIZER
#

lr_schedule = optax.warmup_cosine_decay_schedule(
    init_value=1e-7,
    peak_value=LEARNING_RATE,
    warmup_steps=4000,
    decay_steps=MAX_STEPS,
    end_value=1e-6
)

optimizer = optax.chain(
    optax.clip_by_global_norm(0.5), # Tighter clipping for autoregressive stability
    optax.adamw(
        learning_rate=lr_schedule,
        weight_decay=2e-3, # Increased weight decay for OOD robustness
        b1=0.9,
        b2=0.999
    )
)
```

F.4 Solution $\texttt{sts}_{445}$

This solution is explicitly hierarchical, in three stages that mirror the circuit. The ePT stage applies a rule shared across all eight units, on second-order interaction features of past state, visual channel and bout, with a per-unit output gain and gated leaky integration. The lPT stage is a two-layer per-neuron projection with the connectivity mask enforced in the einsum and no recurrence. The command stage integrates its own past state with the newly computed lPT activity.

```python
import jax
import jax.numpy as jnp
import flax.linen as nn
import optax

#
# CONSTANTS
#

BATCH_SIZE = 512
MAX_STEPS = 500_000
SCHEDULED_SAMPLING_START_PROB = 1.0
SCHEDULED_SAMPLING_FINAL_PROB = 0.0
SCHEDULED_SAMPLING_DECAY_STEPS = 400_000
EARLY_STOP_PATIENCE = 1500
MIN_DELTA = 1e-10
LEARNING_RATE = 1e-3

# NOTE: Connectivity from EPT (cols) to LPT (rows)
CONNECTIVITY = [
 [False, True, True, True, True, True, True, True],
 [False, True, True, True, True, False, True, False],
 [True, True, True, False, True, True, True, True],
 [True, True, True, True, True, False, True, False],
 [True, False, True, True, True, False, False, False],
 [False, False, True, False, True, False, False, False],
 [False, True, True, False, False, False, False, True],
 [False, True, True, True, False, True, True, True],
 [False, True, True, True, False, False, True, True],
 [False, False, False, False, True, True, False, False],
 [False, True, True, True, True, True, False, True],
 [False, True, True, True, True, True, False, True],
 [True, True, True, True, False, True, True, True],
 [True, False, True, False, False, True, True, True],
 [True, True, True, True, True, True, True, False],
 [True, False, True, False, True, True, True, True],
 [True, False, False, False, True, False, True, True],
 [True, False, False, False, False, False, True, False],
 [False, False, False, True, False, True, True, False],
 [False, True, True, True, False, True, True, True],
 [False, False, True, True, False, True, True, True],
 [True, True, False, False, False, False, False, False],
 [True, True, False, True, False, True, True, True],
 [True, True, False, True, False, True, True, True]]

#
# MODEL
#

class Candidate(nn.Module):
    """Hierarchical Neural Dynamic Model with Sparse Connectivity."""

    @nn.compact
    def __call__(self, past_pred, cov_t, train=False):
        B = past_pred.shape[0]

        # 1. Decomposition
        ept_past = past_pred[:, 0:8] # (B, 8)
        ahb_past = past_pred[:, 32:36] # (B, 4)
        bout_state = cov_t[:, 0:1] # (B, 1)
        visual_in = cov_t[:, 1:9] # (B, 8)

        # 2. EPT Update (Neurons 0-7)
        # Shared dynamics rule across all 8 neurons for OOD robustness.
        # Inputs: past state, specific visual channel, and global bout state.
        ept_combined = jnp.stack([ept_past, visual_in, jnp.broadcast_to(bout_state, (B, 8))], axis=-1)

        # Capture second-order interactions [y, v, b, y*v, y*b, v*b]
        ept_feat = jnp.concatenate([
            ept_combined,
            ept_combined[..., [0]] * ept_combined[..., [1]],
            ept_combined[..., [0]] * ept_combined[..., [2]],
            ept_combined[..., [1]] * ept_combined[..., [2]]
        ], axis=-1) # (B, 8, 6)

        # Shared MLP rule
        e = nn.Dense(128)(ept_feat)
```

```
        e = nn.leaky_relu(e)
        e = nn.Dense(128)(e)
        e = nn.leaky_relu(e)

        # Gated leaky integration for autoregressive stability
        ept_gate = nn.sigmoid(nn.Dense(1)(e)).squeeze(-1)
        ept_candidate = nn.sigmoid(nn.Dense(1)(e)).squeeze(-1)

        # Per-neuron individual gain/bias to capture phenotypic variation
        ept_gain = self.param('ept_gain', nn.initializers.ones, (8,))
        ept_out = (1.0 - ept_gate) * ept_past + ept_gate * (ept_candidate * ept_gain)

        # 3. LPT Update (Neurons 8-31)
        # Feed-forward mapping from current EPT, enforcing sparse CONNECTIVITY.
        mask = jnp.array(CONNECTIVITY) # (24 LPT rows, 8 EPT cols)

        # Two-layer sparse projection
        lpt_w1 = self.param('lpt_w1', nn.initializers.glorot_uniform(), (24, 8, 32))
        lpt_b1 = self.param('lpt_b1', nn.initializers.zeros, (24, 32))

        # Each LPT neuron integrates its subset of EPT inputs into a latent space
        lpt_h = jnp.einsum('bi,nik->bnk', ept_out, lpt_w1 * mask[..., None]) + lpt_b1
        lpt_h = nn.leaky_relu(lpt_h)

        lpt_w2 = self.param('lpt_w2', nn.initializers.glorot_uniform(), (24, 32, 1))
        lpt_b2 = self.param('lpt_b2', nn.initializers.zeros, (24,))
        lpt_out = jnp.einsum('bnk,nko->bn', lpt_h, lpt_w2) + lpt_b2
        lpt_out = nn.sigmoid(lpt_out)

        # 4. aHB/nMLF Update (Neurons 32-35)
        # Temporal integrators depending on past state and newly computed LPT.
        ahb_in = jnp.concatenate([ahb_past, lpt_out], axis=-1) # (B, 4 + 24)

        a = nn.Dense(256)(ahb_in)
        a = nn.leaky_relu(a)
        a = nn.Dense(256)(a)
        a = nn.leaky_relu(a)

        ahb_gate = nn.sigmoid(nn.Dense(4)(a))
        ahb_candidate = nn.sigmoid(nn.Dense(4)(a))
        ahb_out = (1.0 - ahb_gate) * ahb_past + ahb_gate * ahb_candidate

        return jnp.concatenate([ept_out, lpt_out, ahb_out], axis=-1)

transition_model = Candidate()

#
# OPTIMIZER
#

lr_schedule = optax.cosine_onecycle_schedule(
    transition_steps=MAX_STEPS,
    peak_value=LEARNING_RATE,
    pct_start=0.1,
    div_factor=10,
    final_div_factor=100
)

optimizer = optax.adamw(
    learning_rate=lr_schedule,
    weight_decay=1e-5
)
```

G Supplementary Figures



Figure S1: **Full search trees for automated model discovery.** Visualization of the complete evolutionary lineage explored by the LLM-guided tree search. Nodes represent executable Python code (candidate models), and edges represent mutations generated by the LLM. Node color indicates the score, with lighter colors representing lower error. The star symbol indicates the best scoring candidate solution. **a.** The unconstrained search tree (`ts`). The best scoring candidate is found after seven generations. **b.** The structure-constrained search tree (`sts`). The best scoring candidate is found after six generations.

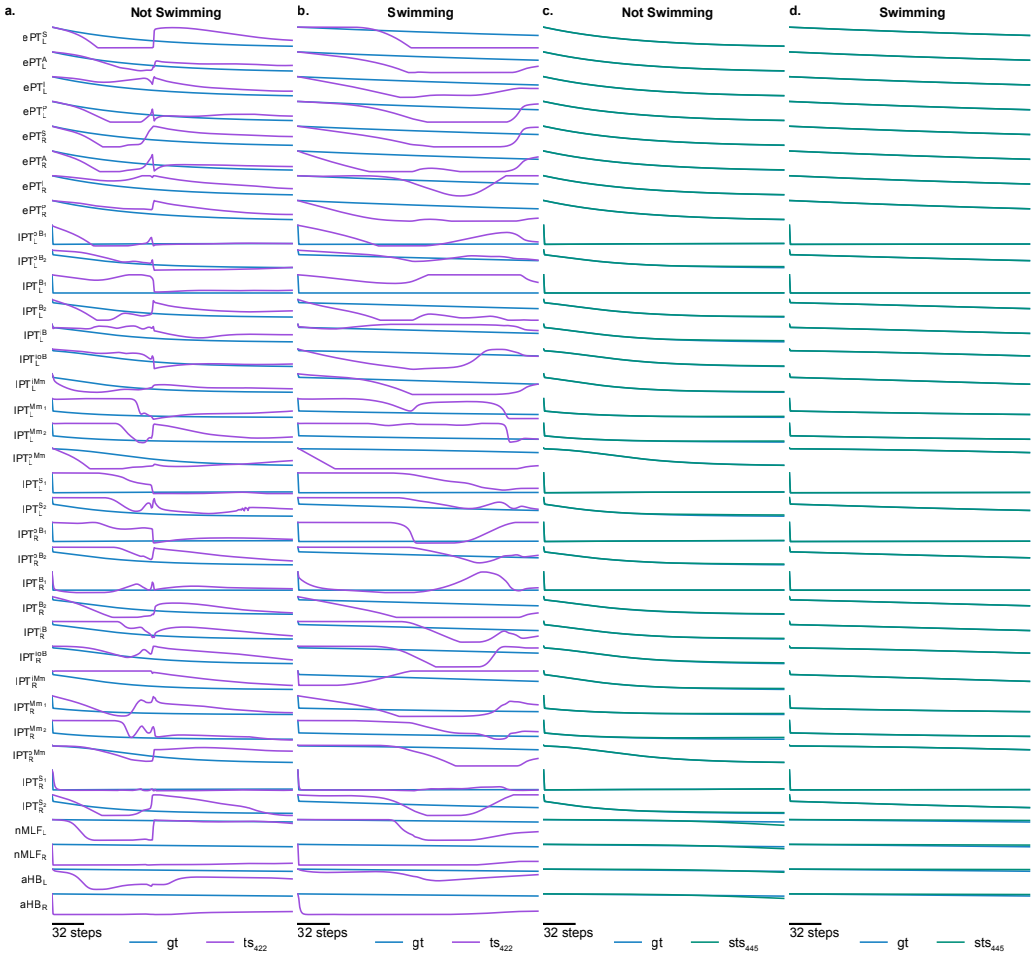

Figure S2: **Impulse response dynamics.** We probe the recovery of dynamic mechanisms by the discovered models by initializing all neurons to a saturated state ($\mathbf{a}_0 = \mathbf{1}$) and evolving the system for 256 steps under "Not Swimming" (Rest) and "Swimming" conditions. **a, b.** $\texttt{ts}_{422}$ (purple) exhibits erroneous oscillatory behavior, deviating significantly from the ground truth (blue). **c, d.** $\texttt{sts}_{445}$ (green) faithfully reproduces the ground truth state-dependent decay profiles that distinguish swimming from resting dynamics.

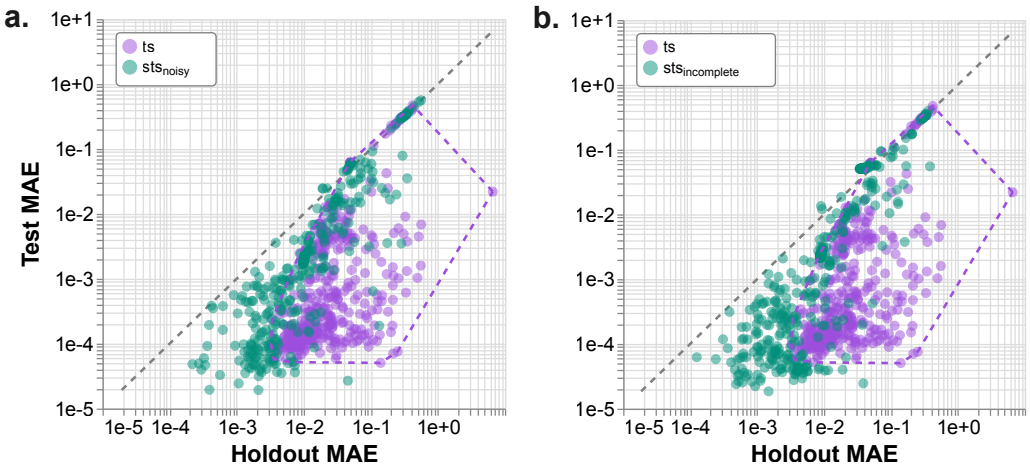

Figure S3: **Robustness to imperfect structural priors. a.** Comparison between in-distribution (Test MAE) versus out-of-distribution stimuli (Holdout MAE) performance for models discovered using a noisy connectome prior ($\texttt{sts}_{\texttt{noisy}}$; and `ts` for reference). **b.** The same comparison for models discovered using an incomplete connectome prior ($\texttt{sts}_{\texttt{incomplete}}$). In both cases, the structure-constrained searches cluster closer to the diagonal compared to unconstrained tree search models (`ts`). Dotted purple line marks the convex hull for `ts` solutions.

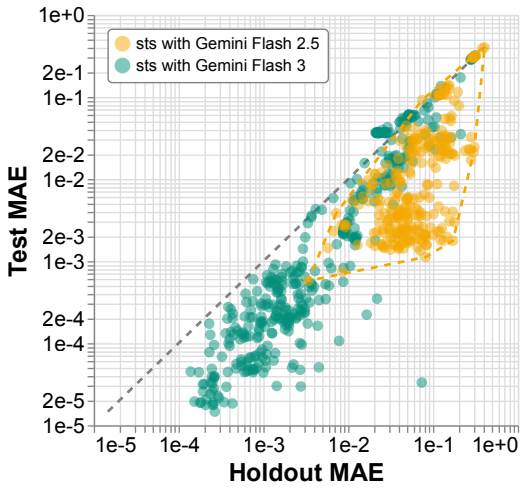

Figure S4: **Impact of the language model backbone on automated discovery.** Performance of models discovered via structure-constrained tree search (sts) using Gemini 2.5 Flash compared to Gemini 3 Flash. While incorporating structural priors with Gemini 3 Flash successfully guided the search toward generalizable models, an identical search utilizing the older Gemini 2.5 Flash model failed to discover solutions with comparable performance. This ablation highlights that providing a structural prior is not sufficient on its own; the success of the discovery process critically depends on the underlying language model. The dotted orange line marks the convex hull of the Gemini 2.5 Flash solutions.

H Supplementary Tables

Table S1: **Performance comparison between best-scoring tree search solutions.** In addition to predictive accuracy (MAE), we report mechanistic verification metrics defined in App. C: the Frobenius norm of the effective connectivity mismatch ($\mathcal{L}_{\text{Jac}}$) and the impulse response error ($\mathcal{L}_{\text{IR}}$) for both resting and swimming states. The structure-constrained solution ($\texttt{sts}_{445}$) demonstrates superior mechanistic fidelity compared to the unconstrained one ($\texttt{ts}_{422}$).

	$\texttt{ts}_{422}$	$\texttt{sts}_{445}$
Test MAE ↓	0.00005	**0.00001**
Holdout MAE ↓	0.13995	**0.00027**
$\mathcal{L}_{\text{Jac}}$ ↓	8.33	**0.26**
$\mathcal{L}_{\text{IR}}^{\text{rest}}$ ↓	0.27	**0.04**
$\mathcal{L}_{\text{IR}}^{\text{swim}}$ ↓	0.29	**0.04**

Table S2: **Population statistics (median [IQR]) for the top 50 highest-scoring models of each tree search.** Aggregated metrics across models discovered during the unconstrained (ts) and structure-constrained (sts) tree searches. The constrained search consistently yields models with lower Jacobian error ($\mathcal{L}_{\text{Jac}}$) and more accurate dynamics ($\mathcal{L}_{\text{IR}}$), indicating that the structural prior effectively regularizes the search space towards higher mechanistic fidelity.

	ts	sts
Test MAE ↓	0.00009 [0.00008, 0.00009]	**0.00004 [0.00002, 0.00005]**
Holdout MAE ↓	0.00833 [0.00636, 0.02349]	**0.00030 [0.00025, 0.00073]**
$\mathcal{L}_{\text{Jac}}$ ↓	7.58 [7.55, 7.73]	**0.34 [0.23, 0.45]**
$\mathcal{L}_{\text{IR}}^{\text{rest}}$ ↓	0.14 [0.10, 0.21]	**0.03 [0.02, 0.04]**
$\mathcal{L}_{\text{IR}}^{\text{swim}}$ ↓	0.12 [0.09, 0.18]	**0.03 [0.02, 0.04]**

Table S3: **Population statistics (median [IQR]) for models with imperfect structural priors.** Aggregated metrics across models discovered during tree searches using a noisy connectome ($\texttt{sts}_{\texttt{noisy}}$) and an incomplete connectome ($\texttt{sts}_{\texttt{incomplete}}$). Despite the degraded priors, the search robustly identifies mechanistic models.

	$\texttt{sts}_{\texttt{noisy}}$	$\texttt{sts}_{\texttt{incomplete}}$
Test MAE ↓	0.00005 [0.00004, 0.00006]	0.00004 [0.00003, 0.00004]
Holdout MAE ↓	0.00212 [0.00153, 0.00352]	0.00363 [0.00104, 0.00482]
$\mathcal{L}_{\text{Jac}}$ ↓	0.99 [0.55, 2.59]	1.00 [0.58, 1.83]
$\mathcal{L}_{\text{IR}}^{\text{rest}}$ ↓	0.09 [0.04, 0.27]	0.08 [0.05, 0.14]
$\mathcal{L}_{\text{IR}}^{\text{swim}}$ ↓	0.09 [0.04, 0.25]	0.08 [0.05, 0.15]

Table S4: **Population statistics (median [IQR]) for models discovered under observation noise.** Aggregated mechanistic recovery metrics across the top 50 models discovered via structure-constrained tree search (sts) at varying levels of AR(1) observation noise. Noise levels are expressed as a percentage of the per-unit standard deviation.

	$\texttt{sts}_{\texttt{AR 2\%}}$	$\texttt{sts}_{\texttt{AR 5\%}}$	$\texttt{sts}_{\texttt{AR 10\%}}$	$\texttt{sts}_{\texttt{AR 20\%}}$
$\mathcal{L}_{\text{Jac}}$ ↓	1.51 [1.18, 3.10]	3.20 [2.88, 3.55]	4.75 [4.25, 5.56]	5.85 [5.36, 6.74]
$\mathcal{L}_{\text{IR}}$ ↓	0.08 [0.04, 0.26]	0.11 [0.08, 0.16]	0.14 [0.09, 0.19]	0.32 [0.18, 0.41]
$\mathcal{L}_{\text{IR}}^{\text{rest}}$ ↓	0.08 [0.03, 0.26]	0.12 [0.08, 0.16]	0.14 [0.08, 0.19]	0.31 [0.17, 0.43]
$\mathcal{L}_{\text{IR}}^{\text{swim}}$ ↓	0.07 [0.04, 0.26]	0.10 [0.08, 0.15]	0.14 [0.10, 0.20]	0.34 [0.18, 0.41]

I Broader Impact

Our in silico testbed provides a transparent, verifiable framework for evaluating model discovery strategies, helping to bridge the gap between predictive AI and biologically plausible mechanistic models. By demonstrating that structural priors can guide automated search systems toward faithful system identification, our approach could accelerate the construction of interpretable neural circuit models from real-world data—contributing to a deeper understanding of brain function.

On the negative side, we caution against over-reliance on AI-generated models without rigorous experimental verification. Our results show that models achieving high predictive accuracy can exploit statistical shortcuts and fail to capture true mechanisms—a risk that may be harder to detect in real-world settings where ground truth is unavailable. Researchers adopting these methods should ideally complement computational model discovery with targeted experimental perturbations to validate mechanistic claims. Additionally, the computational cost of LLM-guided tree search, while manageable in our setting, could become a barrier to entry when scaled to larger systems.

References

[1] Lennart Ljung. Perspectives on system identification. *Annual Reviews in Control*, 2010. URL https://doi.org/10.1016/j.arcontrol.2009.12.001.

[2] Xiangxiao Liu, Matthew D Loring, Luca Zunino, Kaitlyn E Fouke, François A Longchamp, Alexandre Bernardino, Auke J Ijspeert, and Eva A Naumann. Artificial embodied circuits uncover neural architectures of vertebrate visuomotor behaviors. *Science Robotics*, 2025. URL https://doi.org/10.1126/scirobotics.adv4408.

[3] Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting? In *AAAI Conference on Artificial Intelligence*, 2023. URL https://doi.org/10.1609/aaai.v37i9.26317.

[4] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

[5] Abhimanyu Das, Weihao Kong, Andrew Leach, Shaan Mathur, Rajat Sen, and Rose Yu. Long-term forecasting with TiDE. *Transactions on Machine Learning Research*, 2023. URL https://openreview.net/forum?id=pCbC3aQB5W.

[6] Si-An Chen, Chun-Liang Li, Nate Yoder, Sercan O Arik, and Tomas Pfister. TSMixer: An all-MLP architecture for time series forecasting. *Transactions on Machine Learning Research*, 2023. URL https://openreview.net/forum?id=wbpxTuXgm0.

[7] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 2020. URL https://doi.org/10.1016/j.ijforecast.2019.07.001.

[8] Chethan Pandarinath, Daniel J O'Shea, Jace Collins, Rafal Jozefowicz, Sergey D Stavisky, Jonathan C Kao, Eric M Trautmann, Matthew T Kaufman, Stephen I Ryu, Leigh R Hochberg, Jaimie M Henderson, Krishna V Shenoy, L F Abbott, and David Sussillo. Inferring single-trial neural population dynamics using sequential auto-encoders. *Nature methods*, 2018. URL https://doi.org/10.1038/s41592-018-0109-9.

[9] Jan-Matthis Lueckmann, Alexander Immer, Alex Bo-Yuan Chen, Peter H. Li, Mariela D Petkova, Nirmala A Iyer, Luuk Willem Hesselink, Aparna Dev, Gudrun Ihrke, Woohyun Park, Alyson Petruncio, Aubrey Weigel, Wyatt Korff, Florian Engert, Jeff Lichtman, Misha Ahrens, Michal Januszewski, and Viren Jain. ZAPBench: A benchmark for whole-brain activity prediction in zebrafish. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=oCHsDpyawq.

[10] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 2016. URL https://doi.org/10.1038/nature16961.

[11] Eser Aygün, Anastasiya Belyaeva, Gheorghe Comanici, Marc Coram, Hao Cui, Jake Garrison, Renee Johnston, Anton Kast, Cory Y. McLean, Peter Norgaard, Zahra Shamsi, David Smalling, James Thompson, Subhashini Venugopalan, Brian P. Williams, Chujun He, Sarah Martinson, Martyna Plomecka, Lai Wei, Yuchen Zhou, Qian-Ze Zhu, Matthew Abraham, Erica Brand, Anna Bulanova, Jeffrey A. Cardille, Chris Co, Scott Ellsworth, Grace Joseph, Malcolm Kane, Ryan Krueger, Johan Kartiwa, Dan Liebling, Jan-Matthis Lueckmann, Paul Raccuglia, Xuefei Wang, Katherine Chou, James Manyika, Yossi Matias, John C. Platt, Lizzie Dorfman, Shibl Mourad, and Michael P. Brenner. An AI system to help scientists write expert-level empirical software. *Nature*, 2026. URL https://doi.org/10.1038/s41586-026-10658-6.

[12] Google. Gemini 3 Flash: Frontier intelligence built for speed, December 17 2025. URL https://blog.google/products-and-platforms/products/gemini/gemini-3-flash/. The Keyword.

[13] Google. Developers can now start building with Gemini 2.5 Flash, April 17 2025. URL https://blog.google/products-and-platforms/products/gemini/gemini-2-5-flash-preview/. The Keyword.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction enumerate five specific contributions. Each is directly supported by our results and discussion.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Limitations are explicitly stated as part of the discussion.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: The paper does not present formal theorems or proofs; our contribution is empirical rather than theoretical.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The paper provides: (1) the complete ground-truth neural circuit equations and all numerical parameter values (Sec. 3.1 and App. B); (2) the system identification task definition including data splits, prediction horizon, and evaluation metrics (Sec. 3.2); (3) full hyperparameters for all baseline models (App. D); (4) the tree search algorithm, prompts, and implementation details (App. E); and (5) code for the discovered models $\texttt{ts}_{001}$, $\texttt{ts}_{422}$, and $\texttt{sts}_{445}$ (App. F). The underlying simulator [5] is openly available (`https://ponyo.epfl.ch/proj/zebrafish/simzfish`).

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility.

In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: As stated in Sec. 3.2, code (including the JAX-translated neural circuit model) and datasets are available at `https://in-silico-sysid.web.app`. The underlying simulator is already open source [5] (`https://ponyo.epfl.ch/proj/zebrafish/simzfish`).

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Data splits are described in Sec. 3.2 (70% train, 10% val, 20% test, with a fourth condition as OOD holdout). Hyperparameters for all baselines (optimizer, learning rate, weight decay, architecture details) are in App. D. Tree search details including the LLM backbone, maximum nodes, and runtime limits are in App. E.2. Code listings for best-scoring solutions (including optimizers) are in App. F.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

Justification: The paper reports population statistics (median and interquartile range) over the top 50 models from each tree search in Tab. S2, capturing variability across the search process. The main comparison between specific models (ts_{422} vs. sts_{445}) in Tab. S1 shows differences of more than an order of magnitude on key metrics, and the population-level results in Tab. S2 and Tab. S3 confirm these findings are robust.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: App. E.2 specifies the GPU type (NVIDIA T4), the runtime limit per candidate solution (12 hours), and the maximum number of search nodes (500). The tree search used Gemini 3 Flash as the backbone LLM.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The paper uses synthetic data generated from an open-source neuromechanical simulator (simZFish). No human subjects or animal experiments are involved. The work does not raise concerns related to dual use, surveillance, or discrimination.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.

- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: App. I discusses both potential positive and negative broader impacts.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The released artifacts will consist of synthetic activity data from an open-source simulator and transition models for the circuit, which pose no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The paper cites all external assets and baseline model architectures. simZFish is open source, released under Apache 2.0; `https://ponyo.epfl.ch/proj/zebrafish/simzfish`.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The paper introduces a new *in silico* testbed (dataset, JAX-based neural circuit model, system identification task, and metrics). The model and dataset generation process is documented in Sec. 3.1 and App. B. Code and data are available at `https://in-silico-sysid.web.app`.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing or research with human subjects. All experiments use synthetic data generated by a neuromechanical simulator.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve human subjects research. All experiments are computational, using synthetic data from an *in silico* simulator.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLMs are a core methodological component: the paper uses Gemini 3 and 2.5 Flash as the backbone for LLM-guided tree search to generate and mutate candidate neural architectures (Sec. 3.2, App. E). The specific model, its role, and the prompts provided to it are fully disclosed.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.