EDEN: Emergent Dynamics in Evolutionary Neural-networks for Robust Continuous Control

Chi Zhang[1]* **Jinge Li**[1,2]* **Yifei Wang**[1] **Lei Wang**[1] **Tianyi Qian**[1]†

[1]Qiyuan Laboratory, Beijing, China

[2]School of Medical Science and Engineering, Beijing Institute of Technology, Beijing, China

zhangchi@qiyuanlab.com qiantianyi@qiyuanlab.com

Abstract

Biological neural systems leverage pronounced neuronal heterogeneity and diverse firing patterns to generate rich non-linear dynamics—features that are largely abstracted away in traditional artificial neural networks (ANNs) and simplified spiking neural networks (SNNs). Although detailed differential equation-based neural dynamics models offer profound biological plausibility and expressive capacity, the discontinuity of spike generation and complex state evolution processes pose significant challenges for direct gradient-based optimization, severely limiting their application in complex functional tasks. To break this bottleneck, we introduce EDEN (Emergent Dynamics in Evolutionary Neural-networks), a neural dynamics model training framework inspired by natural evolutionary processes. Built upon a continuous-time network of heterogeneous Izhikevich neurons, EDEN seamlessly integrates efficient parallel neural dynamics simulation with advanced evolutionary strategies, enabling the joint optimization of intrinsic single-neuron dynamics parameters and network synaptic connections entirely without reliance on backpropagation. Comprehensive evaluations across multiple MuJoCo continuous motor control tasks demonstrate that EDEN achieves robust control performance comparable to mainstream deep reinforcement learning baselines. In-depth neurodynamic analysis further reveals that the trained EDEN models exhibit high biological interpretability, characterized by diverse single-neuron functional differentiation and stable low-dimensional population dynamics manifolds. Neurobiology-informed simulation experiment further shows that EDEN exhibits strong representational alignment with macaque M1 neural data. This study demonstrates that gradient-free evolutionary optimization is an effective pathway to overcome the training bottlenecks of complex neural dynamics systems, offering a highly promising paradigm for developing brain-inspired computational models that combine functional execution capability with biological plausibility.

1 Introduction

Biological organisms exhibit remarkable adaptability by continuously interacting with non-stationary, high-dimensional environments under selective evolutionary pressure. A central substrate of this adaptability is the nervous system, which leverages complex neural dynamics to transform transient sensory experiences into precise adaptive behaviors. Early computational frameworks therefore sought to capture these phenomena through biophysically grounded models, focusing on membrane-potential evolution, synaptic conductance, and spike-generation mechanisms [1]. However, the significant computational cost of these dynamical models prompted the development of simplified artificial

*Equal contribution.

†Corresponding author: Tianyi Qian (qiantianyi@qiyuanlab.com).

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

neural networks (ANNs) [2], which eventually emerged as the dominant paradigm, facilitated by the advent of backpropagation-based training [3, 4]. While ANNs have achieved unprecedented success in perception, representation learning, and control, they typically formalize computation as static input–output mappings, and often rely on large-scale supervised optimization [5]. This abstraction overlooks the rich temporal hierarchies and continuous-time properties intrinsic to biological substrates. A key challenge is therefore to develop neural network models that preserve biologically inspired neurodynamic properties while satisfying the functional requirements of complex closed-loop control environments.

Neurodynamic modelling has re-emerged as a promising route for linking biological mechanisms to intelligent behavior. Balanced recurrent networks show that neural interactions can generate irregular yet stable population dynamics [6, 7], while spiking neural networks (SNNs) leverage event-driven communication to perform sophisticated temporal computation [8]. Recent advances in surrogate gradients, adaptive neuronal dynamics, and local learning rules have significantly enhanced SNN's trainability [9–12]. Nevertheless, most neurodynamic models still rely on simplified leaky integrate-and-fire (LIF) neurons, surrogate derivatives, or explicit structural constraints to guarantee stability and scalability, often at the expense of intrinsic dynamical expressiveness.

In contrast, biological neurons exhibit pronounced heterogeneity and diverse firing patterns. The Izhikevich model provides a compact and efficient formulation capable of reproducing multiple biologically observed firing behaviors [13, 14]. However, optimizing heterogeneous Izhikevich networks remains challenging due to nonlinear state evolution, long-range temporal dependencies, and non-differentiable spikes. Inspired by natural evolution, evolution strategies (ES) provide a gradient-free alternative through population sampling and fitness-based updates [15, 16], yet directly applying them to high-dimensional neuronal and synaptic parameter spaces remains computationally demanding and prone to inefficient exploration. Thus, functionalizing biologically expressive neurodynamic systems requires optimization methods that preserve rich neural dynamics while ensuring computational scalability.

To address this challenge, we propose **EDEN** (**Emergent Dynamics in Evolutionary Neural-networks**), an evolutionary neurodynamic framework for robust continuous control. EDEN grounds policy computation in continuous-time networks composed of heterogeneous Izhikevich neurons, where adaptive behavior emerges from the interaction among intrinsic neuronal dynamics, recurrent synaptic connectivity, and environment-driven state evolution. Rather than simplifying neuronal properties or imposing restrictive structures to ease optimization, EDEN jointly optimizes neuron-specific intrinsic parameters and synaptic weights. Building on the high-performance simulation framework ENLARGE [17], we redesign data structures for heterogeneous neurons and synapses and integrate parallel evolution strategies to improve exploration efficiency over complex dynamical landscapes. Evaluated across multiple MuJoCo continuous-control tasks, EDEN evolves neurodynamic controllers that achieve robust performance competitive with state-of-the-art deep reinforcement-learning baselines. Moreover, the resulting models exhibit interpretable single-neuron differentiation and stable low-dimensional population dynamics. Through neurobiology-informed experiments and comparisons with M1 neural recordings, EDEN extends its biological consistency from intrinsic neural dynamics to motor control, offering insights into the neural mechanisms underlying evolved behaviors.

Contributions

EDEN framework. This work introduces EDEN, a heterogeneous Izhikevich-neuron framework that represents closed-loop policies through continuous-time neural dynamics.

Evolutionary training. A parallel, gradient-free optimization pipeline is developed to enable the jointly evolving intrinsic neuronal parameters and synaptic weights, bypassing the differentiable substrates. Empirical evaluations demonstrate that EDEN achieves robust performance on MuJoCo tasks.

Emergent biologically consistent control. EDEN exhibits biologically interpretable single-neuron differentiation and stable low-dimensional population dynamics, together with task-related organization aligned with macaque M1 population representations.

Overall, EDEN suggests an alternative paradigm towards brain-inspired intelligence: functional closed-loop behavior can emerge from heterogeneous neural dynamics shaped by evolutionary optimization, without relying on static mapping-based computation or explicit gradient calculation.

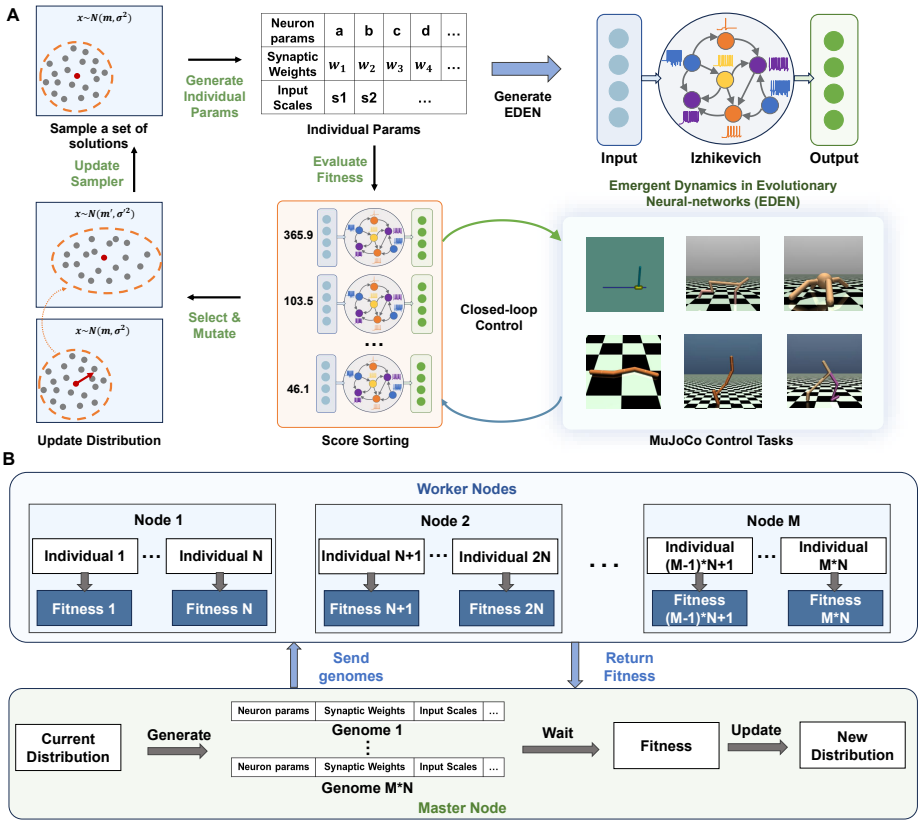

Figure 1: **The EDEN framework.** (A) Neurodynamic optimization workflow. Successive generations of heterogeneous dynamical network parameters are sampled from the parameter distribution of the preceding population. Each individual within a generation is instantiated as a neurodynamic model that evolves over time through continuous closed-loop interaction with MuJoCo environments. The resulting task returns are assigned as fitness values, which are subsequently used to update the sampling distribution for the following generation. (B) Parallel evolution workflow. A master node generates the complete set of neurodynamic models for the current generation and dispatches individual models to multiple worker nodes according to computational load. These nodes concurrently execute neurodynamic simulations and environment interactions. Upon completion, fitness values are aggregated by the master node to update the parameters for the subsequent evolutionary cycle.

2 Related Works

Neurodynamic Models Neurodynamic models describe neural computation through continuous-time state evolution and provide a foundation for understanding electrophysiological mechanisms, excitation–inhibition balance, and population dynamics [1, 6, 7]. Recent work has extended spiking neural networks (SNNs), recurrent spiking neural networks (RSNNs), and continuous-time recurrent models from biological activity modelling to temporal processing, reinforcement learning, and robotic closed-loop control. Spike-based communication, adaptive thresholds, population coding, recurrent connectivity, and biologically inspired circuit structures have all been shown to support functional control [18–22]. For example, population-coded spiking actors achieve ANN-comparable control performance with improved neuromorphic efficiency, while TBC-HRL uses timed subgoal scheduling and a *C. elegans*-inspired circuit to improve hierarchical control stability and interpretability [18, 22]. However, these methods mainly embed spiking or bionic components into existing deep reinforcement-learning frameworks. In contrast, we ask whether functional behavior can emerge directly from highly heterogeneous neurodynamic networks, which also aligns with recent neuroscience-oriented studies emphasizing the relationship between neural variability and low-dimensional latent dynamics [23–26].

Training Neurodynamic Systems Training complex neurodynamic systems remains challenge because discrete spikes, continuous state evolution, and recurrent connectivity jointly create challenging underlying landscapes. Existing approaches improve trainability through surrogate gradients, local plasticity, online training, and structural constraints. Surrogate-gradient methods approximate the non-differentiable spike function during backpropagation, enabling deep SNNs to be optimized with standard gradient-based methods [9, 11, 27]. In continuous-control reinforcement learning, surrogate-gradient and hybrid ANN–SNN actor–critic methods have enabled SNN policies to bridge the gap with ANN-based counterparts [18, 28, 29]. Nevertheless, these methods often depend on simplified neuron models, surrogate derivatives, or restrictive architectural assumptions. Training biologically inspired networks with heterogeneous neuronal properties, diverse firing patterns, and strongly nonlinear dynamics without backpropagation therefore remains an open challenge.

Evolutionary Strategies ES provide a gradient-free black-box optimization paradigm for non-differentiable dynamical systems through population sampling and fitness evaluation [16, 30, 31]. In reinforcement learning, ES scales naturally through parallel population evaluation and reduces reliance on long-horizon credit assignment [16, 32]. Evolutionary methods have also been used for SNN and RSNN policy optimization; for instance, Evolving Connectivity applies NES to optimize sparse Boolean recurrent connectivity for robotic locomotion [20]. However, most evolutionary SNN studies reduce search difficulty by compressing connectivity, restricting weight parameterization, enforcing sparsity, or fixing neuronal dynamics. EDEN instead directly optimizes a high-dimensional parameter space composed of heterogeneous Izhikevich neurons and synaptic connections, examining whether adaptive control can evolve from biologically grounded neural dynamics.

3 Methods

As illustrated in Fig. 1, EDEN facilitates closed-loop control through interaction with MuJoCo. Within the evolutionary framework, each candidate individual is instantiated as a neurodynamic network comprising 64 heterogeneous Izhikevich neurons. The parameter space for each individual encompasses the intrinsic excitability parameters of each constituent neuron alongside the synaptic weight matrix governing the directed connections between them. This formulation ensures that the evolutionary process jointly optimizes the fine-grained neuronal firing characteristics and the emergent population-level connectivity.

During each environment step, the current observation is encoded into specialized input currents and send to the neural population. The heterogeneous Izhikevich neurons then undergo evolution over multiple internal simulation sub-steps, capturing fine-grained temporal dynamics before the resulting population activity is decoded into a continuous action vector. This action prompts the environment to return the subsequent observation and a scalar reward. The cumulative return over a complete episode defines the fitness of the candidate, providing the objective signal for updating the evolutionary distribution and generating the succeeding population of neurodynamic networks.

3.1 Heterogeneous Izhikevich Neuron Model

The fundamental computational building blocks of EDEN are Izhikevich neurons, which facilitate the replication of diverse, biologically observed firing regimes with high computational efficiency[13, 14]. The state of the i-th neuron is described by its membrane potential v_i and recovery variable u_i:

$$\frac{dv_i}{dt} = 0.04v_i^2 + 5v_i + 140 - u_i + I_i, \tag{1}$$

$$\frac{du_i}{dt} = a_i(b_i v_i - u_i). \tag{2}$$

When the membrane potential reaches the firing threshold, the neuron emits a spike and is reset according to

$$\text{if } v_i \geq 30\,\text{mV}, \quad v_i \leftarrow c_i, \quad u_i \leftarrow u_i + d_i. \tag{3}$$

Here, (a_i, b_i, c_i, d_i) are trainable heterogeneous neuronal parameters that control the recovery time scale, membrane-potential sensitivity, reset potential, and post-spike recovery strength, respectively. In EDEN, each neuron is assigned an independent and trainable set of intrinsic parameters, enabling the network to contain diverse dynamical units with distinct firing properties.

The total input current of neuron i consists of observation-driven current, recurrent synaptic current, and stochastic perturbation:

$$I_i(t) = I_i^{\text{obs}}(t) + I_i^{\text{syn}}(t) + \xi_i(t). \tag{4}$$

The observation-driven current is obtained by linearly weighting the environment observation. The stochastic perturbation is sampled as $\xi_i(t) \sim \mathcal{N}(0, \sigma_\xi^2)$, where σ_ξ controls the noise intensity. The recurrent postsynaptic current is given by

$$I_i^{\text{syn}}[t] = \sum_j w_{ij} \sum_f \delta\left[t - t_j^{(f)}\right], \tag{5}$$

where w_{ij} denotes the synaptic weight from presynaptic neuron j to postsynaptic neuron i, $t_j^{(f)}$ is the discrete time index of the f-th spike emitted by presynaptic neuron j, and $\delta[\cdot]$ denotes a discrete impulse function. Positive and negative values of w_{ij} correspond to excitatory and inhibitory synaptic effects, respectively, while the magnitude of w_{ij} controls the strength of the postsynaptic current induced by presynaptic spikes.

3.2 Dynamic Interaction between EDEN and Mojoco

An EDEN individual consists of three components: an input mapping layer, a neurodynamic model, and an output decoding layer. The input mapping layer converts the environment observation o_t into external input currents for Izhikevich neurons. The neurodynamic model serves as the hidden dynamical layer, which is composed of a heterogeneous neuronal population with fully connected recurrent synapses. The output decoder maps the resulting neural population activity to continuous actions in the environment.

At each MuJoCo control step, the environment observation o_t is encoded into observation-driven input currents through a trainable affine mapping,

$$I_t^{\text{obs}} = \gamma_1(W_{\text{in}} o_t + b_{\text{in}}), \tag{6}$$

which drives the recurrent Izhikevich population during the internal neurodynamic simulation. This direct-current interface is motivated by intracellular electrophysiology, where constant, fluctuating, or synaptic-like currents are injected to characterize membrane-potential and spiking responses [33, 34]. After K internal simulation steps, the spike activity is aggregated into a population firing-rate vector,

$$r_t = \frac{1}{K} \sum_{n=1}^{K} z[n], \tag{7}$$

where $z[n] \in \{0,1\}^N$ denotes the spike vector at internal step n. Windowed spike counts and firing-rate estimates are standard descriptions of single-neuron and population activity in neurophysiological analysis [35–37]. The continuous action applied to MuJoCo is then decoded as

$$a_t = a_{\max} \tanh\left(\gamma_2 W_{\text{out}} r_t + b_{\text{out}}\right), \tag{8}$$

where $a_{\max}$ denotes the action bound of the environment. Thus, EDEN forms a closed-loop controller by mapping physical observations to neural input currents and decoding recurrent population activity into continuous motor commands.

Each individual θ contains a complete set of evolvable parameters:

$$\theta = \{W_{\text{in}}, b_{\text{in}}, \{a_i, b_i, c_i, d_i\}_{i=1}^N, W_{\text{rec}}, W_{\text{out}}, b_{\text{out}}, \gamma_1, \gamma_2\}. \tag{9}$$

Here, $\{a_i, b_i, c_i, d_i\}_{i=1}^N$ are neuron-specific intrinsic dynamical parameters, W_{rec} denotes recurrent synaptic weights, and the remaining parameters define the input and output transformations. If the noise intensity is optimized, σ_ξ is also included in θ; otherwise, it is treated as a fixed simulation hyperparameter. Details on parameters can be found in Appendix.

3.3 Parallel Evolutionary Workflow

At generation g, the evolutionary algorithm maintains a current parameter center m_g. The master node samples perturbations around this center to generate λ candidate individuals:

$$\theta_k = m_g + \sigma \epsilon_k, \quad \epsilon_k \sim \mathcal{N}(0, I), \quad k = 1, \ldots, \lambda. \tag{10}$$

As shown in Fig. 1B, each candidate θ_k is dispatched to a worker node, where the corresponding neurodynamic network is constructed and evaluated through parallel episode rollouts in MuJoCo. The cumulative return $F(\theta_k)$ is used as the fitness of the candidate. After all workers return their fitness values, the master node ranks or normalizes the returns and converts the raw fitness values into utility weights u_k. The parameter center is then updated by the estimated evolutionary search direction:

$$m_{g+1} = m_g + \frac{\eta}{\lambda\sigma} \sum_{k=1}^{\lambda} u_k \epsilon_k. \tag{11}$$

Here, η is the learning rate, σ is the perturbation standard deviation, and λ is the population size. The updated center m_{g+1} is used to sample the next generation of candidate networks. More detailed methodological descriptions are provided in the Appendix.

4 Experiments

Tasks. The performance of EDEN is evaluated across seven standard continuous-control tasks from OpenAI Gym and MuJoCo [38, 39]: InvertedDoublePendulum, Swimmer, Ant, HalfCheetah, Hopper, Walker2d, and Humanoid. These environments serve as benchmarks within the reinforcement learning community [40], encompassing a broad spectrum of control regimes that range from fundamental balancing to complex, high-dimensional multi-joint locomotion. Beyond these seven MuJoCo motor-control benchmarks, we further evaluate EDEN in an eight-direction center-out reaching task to assess the biological plausibility of both its closed-loop control behavior and population-level neural representations through comparison with macaque reaching behavior and primary motor cortex (M1) recordings [41].

Baselines. EDEN is compared against established gradient-based deep reinforcement learning (DRL) algorithms. Specifically, PPO [42] and SAC [43] serve as reference performance standards for gradient-trained ANN controllers. While PPO and SAC utilize standard actor–critic architectures, EDEN operates solely as an actor policy manifested through an evolved neurodynamic network.

Comparative Experiments. To isolate the specific contributions of the proposed neurodynamic architecture, EDEN is compared with the conventional ANN, RNN, the standard SNN, and the homogeneous Izhikevich network under identical evolutionary optimization settings. The standard SNN baseline uses leaky integrate-and-fire (LIF) neurons, whereas the homogeneous Izhikevich network has fixed intrinsic dynamics parameters. In contrast, EDEN employs heterogeneous Izhikevich neurons with trainable intrinsic dynamics. Standard MLP and RNN architectures serve as artificial neural network references: the MLP implements a static feedforward mapping from observations to actions, while the RNN incorporates recurrent hidden states to facilitate temporal processing.

Neurodynamic Analysis. To investigate the internal dynamics of trained EDEN policies, evolved networks are subjected to closed-loop control tasks while simultaneously recording single-neuron membrane potentials and population spike sequences. At the single-neuron level, membrane-potential traces are used to examine whether different neurons develop heterogeneous activity patterns within the same task. At the population level, neural activity during task execution is organized as a time-by-neuron activity matrix. Principal Component Analysis (PCA) [44] is applied to the centered population activity to extract the first three principal components. This dimensionality reduction facilitates the visualization of low-dimensional neural manifolds and population trajectories, enabling a assessment of the underlying neural manifolds that governs adaptive behavior.

Deployment Environment. All training experiments are conducted on a distributed CPU cluster with 40 compute nodes. Each node is equipped with two Intel® Xeon® Platinum 8358 CPUs at 2.60 GHz and 512 GB memory. Candidate EDEN networks are evaluated in parallel on worker nodes, while the master process collects episode returns and updates the evolutionary search distribution.

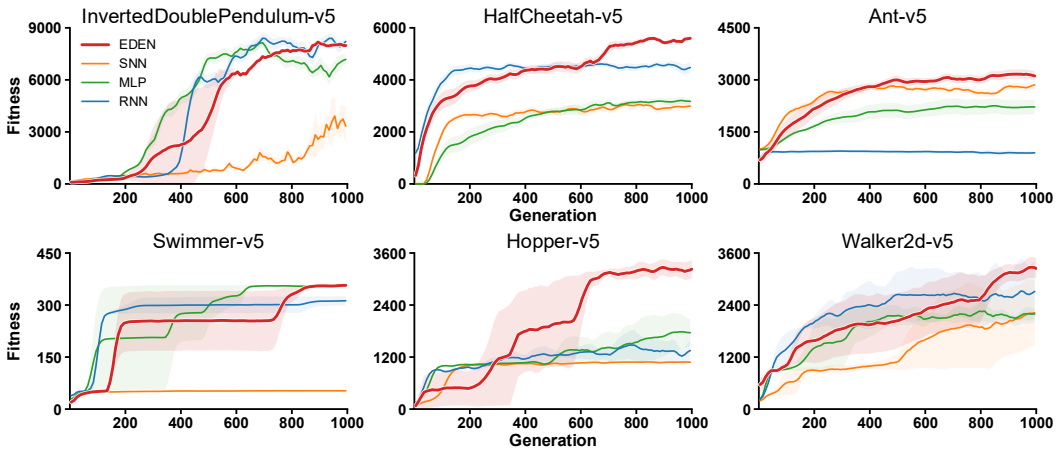

Figure 2: Comparative Learning Curves Across Diverse Framework. The panels illustrate the cumulative returns of ANN, RNN, SNN, and EDEN controllers during the evolutionary optimization process across six control tasks. To ensure a rigorous and controlled comparison, all architectures were trained using an identical Evolution Strategy (ES) framework. Solid lines and shaded regions represent the mean and standard deviation, respectively, calculated over three independent random seeds.

5 Results

5.1 Performance Evaluation

Table 1 reports the performance comparison between EDEN and standard deep reinforcement-learning references. PPO and SAC are used to indicate the performance level of conventional MLP-based controllers in the same MuJoCo environments. We also compare EDEN with a conventional SNN built using LIF neurons and an Izhikevich network in which neuronal intrinsic parameters are fixed. Across multiple tasks, EDEN, using only a single hidden neurodynamic layer with 64 Izhikevich neurons, achieves competitive control performance relative to MLP baselines with two hidden layers of 64 neurons, and also outperforms SNN and homogeneous Izhikevich networks.

Table 1: Performance comparison of SNN, homogeneous Izhikevich networks, and EDEN against MLP-based PPO and SAC across seven MuJoCo continuous-control tasks. PPO and SAC are reference scores reported in the original studies [42, 43]. Results for SNN, homogeneous Izhikevich networks, and EDEN are trained for 1000 evolutionary generations. Bold values indicate the best performance for each task.

Task	PPO-Ref	SAC-Ref	SNN	Homogeneous Izhikevich	EDEN
InvertedDoublePendulum	9231	9359	3329.4 ± 1155.9	3959.7 ± 5065.1	$\mathbf{9359.4} \pm 0.2$
Swimmer	67	44	56.3 ± 0.4	177.8 ± 104.6	$\mathbf{363.7} \pm 1.5$
HalfCheetah	5784	**10400**	3679.9 ± 36.2	6224.7 ± 333.2	6301.9 ± 83.6
Hopper	2609	3370	1161.9 ± 1.5	2103.6 ± 216.5	$\mathbf{3801.2} \pm 50.2$
Walker2d	3588	3740	2883.3 ± 1141.8	2388.4 ± 398.2	$\mathbf{3920.4} \pm 489.3$
Ant	3258	3720	3713.0 ± 74.4	1758.3 ± 811.8	$\mathbf{4053.6} \pm 323.9$
Humanoid	787	5200	1151.7 ± 73.6	6102.8 ± 1118.1	$\mathbf{8720.4} \pm 1078.8$

5.2 Comparative Experiments

To further examine the contribution of the proposed framework, four network configurations (ANN, RNN, SNN, and EDEN) were evaluated under identical evolutionary optimization settings. As shown in Fig. 2, EDEN achieves superior performance across the majority of MuJoCo tasks outperforming MLP, RNN and SNN baselines of comparable scale. The training progress further indicate that EDEN demonstrate a more stable optimization trajectories and higher final rewards across several tasks. In contrast, the alternative networks do not exhibit consistently robust performance over the full set of benchmarks. Additional efficiency and ablation analyses are provided in the Appendix.

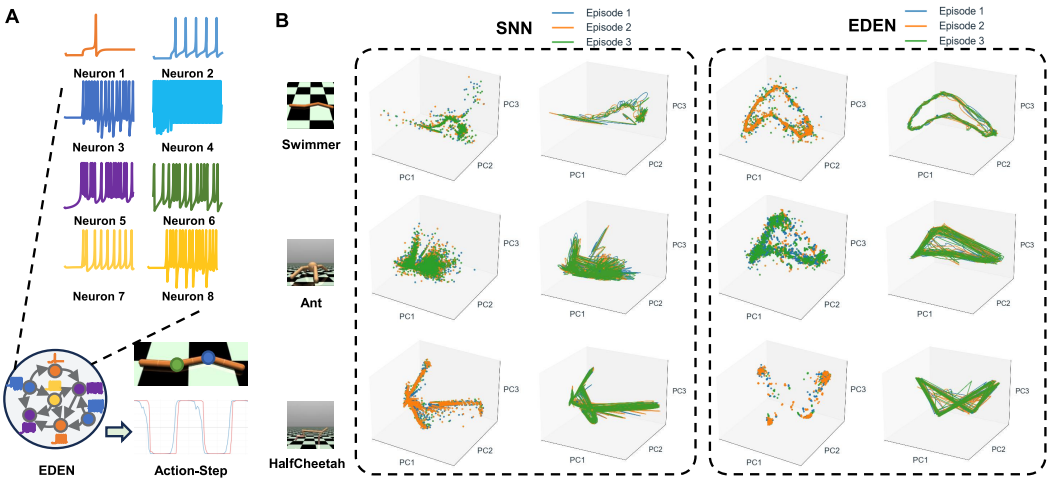

Figure 3: Neurodynamic analysis of EDEN. (A) Heterogeneous Neuronal Firing Regimes. Representative membrane-potential traces from eight selected neurons within the evolved EDEN network exhibit distinct, task-specific firing patterns during the Swimmer task. (B) Low-dimensional Population Trajectories. PCA projections of neural population activity across Swimmer, Ant, and HalfCheetah benchmarks are shown. In contrast to the fixed-parameter SNN baseline, EDEN generates more stable and topologically structured low-dimensional manifolds across multiple episodes.

5.3 Neurodynamic Analysis of EDEN

To further investigate the biological properties emerging from EDEN during closed-loop interactions, the analysis focuses on heterogeneous single-neuron firing regimes and their underlying low-dimensional population dynamics. As illustrated in Fig. 3A, membrane-potential traces from eight representative neurons during the Swimmer task reveal diverse firing characteristics within the same behavioral episode. These patterns, encompassing sustained firing, intermittent activity, and phase-specific activation, demonstrate that evolutionary optimization preserves neuronal diversity rather than collapsing the network into a homogeneous activity state. This suggests that EDEN facilitates the emergence of functionally differentiated neurodynamics tailored to the requirements of the task.

At the population level, PCA of the task-evoked population spike matrix reveals that trained EDEN activity forms stable low-dimensional manifolds, whereas SNN activity is relatively randomly scattered in the high-dimensional state space, as shown in Fig. 3B. Further trajectory analysis shows that population activity of EDEN evolves along structured low-dimensional trajectories with ring-like and saddle-like geometries. These trajectories are consistent with low-dimensional neural population dynamics observed in the motor cortex during rhythmic locomotor behaviors [45]. In rhythmic tasks such as Swimmer and Ant, EDEN-evolved ENNs spontaneously form stable, trial-consistent closed-loop low-dimensional trajectories without explicit programmatic constraints, suggesting that their periodic motor control arises from intrinsic neurodynamic attractors. These results suggest that EDEN not only shapes functional control behavior, but also gives rise to biologically interpretable neural dynamics that are consistent with established neuroscientific principles.

5.4 Biological Consistency Analysis of EDEN

To further assess the biological consistency of EDEN across behavior and neural population activity, we grounded the evaluation in a task derived from an established motor-control experiment in neuroscience. We adopted the classic eight-direction center-out reaching paradigm, in which rhesus macaques control an onscreen cursor using a two-link planar manipulandum. In each trial, the macaque moves the cursor from a central target to one of eight equally spaced peripheral targets and holds it there for 0.5 s [41]. Following this paradigm, we implemented a two-joint planar reaching task in Brax [46], with eight peripheral targets. At the beginning of each episode, the arm is initialized

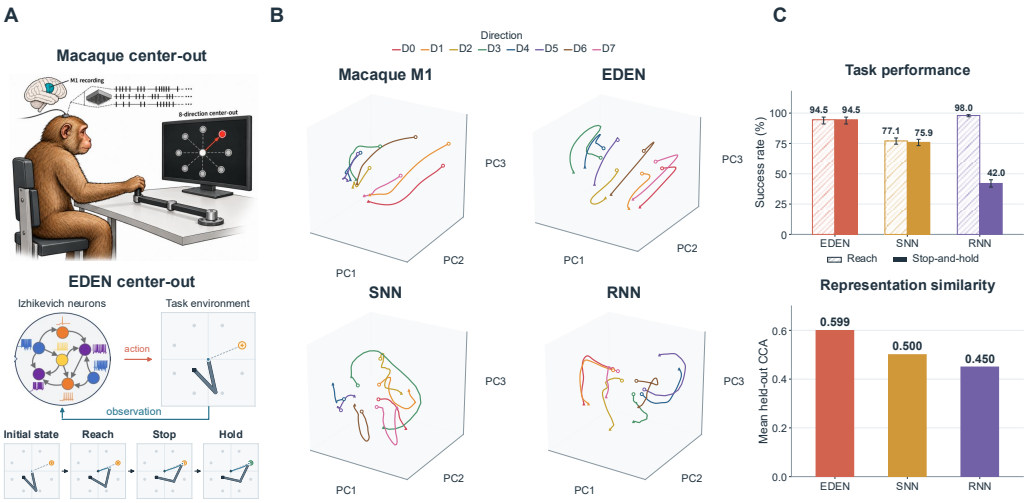

Figure 4: Behavioral and population-representation comparisons between EDEN and macaque neural data. (A) Task setup (adapted from Gallego-Carracedo et al. [41]). Top: the macaque center-out reaching task and M1 neural recording protocol; bottom: the corresponding EDEN simulation task and control procedure. (B) Low-dimensional neural representations. PCA trajectories of population activity from macaque M1, EDEN, SNN, and RNN are shown for the eight reach directions. (C) Quantitative comparisons. Top: success rates under the Reach and Stop-and-hold criteria; bottom: the mean of the first three held-out canonical correlations (CCA) between each model activity and macaque M1 activity.

near the central position and is required to reach the specified target, decelerate, and maintain its endpoint within the target region (Fig. 4A).

We compared EDEN with SNN and RNN controllers in terms of task performance and neural representations, with a publicly available macaque M1 dataset as the biological reference [41]. EDEN achieved consistently high success rates under both the Reach and Stop-and-hold criteria, whereas the SNN and RNN showed either lower overall success or a larger performance gap between the two criteria (Fig. 4C, top). At the population level, PCA trajectories of EDEN activity formed a low-dimensional structure organized by reach direction and evolved systematically with movement progress (Fig. 4B). This direction-dependent temporal organization qualitatively resembled that observed in macaque M1. CCA-based representational analysis further showed that the population activity of EDEN could be linearly aligned with the M1 reference and yielded a higher similarity score than the SNN and RNN representations (Fig. 4C, bottom). Together, these results provide evidence that EDEN exhibits biologically consistent features across multiple scales, from its neuron-level dynamical formulation to population-level organization and closed-loop behavior.

6 Conclusion

This work introduces EDEN, an evolutionary neurodynamic framework inspired by biological evolution, and evaluated it on multiple MuJoCo continuous-control tasks. The results show that EDEN networks composed of heterogeneous Izhikevich neurons and recurrent synaptic connections can evolve functional control policies without relying on backpropagation or surrogate gradients. Beyond task performance, EDEN exhibits biologically interpretable internal dynamics, including diverse single-neuron firing patterns and stable low-dimensional population trajectories. Together, the reaching and M1 comparisons provide descriptive evidence that EDEN's evolved dynamics support task-relevant organization across both behavior and neural population activity. These findings suggest that EDEN can serve not only as a tool for simulating neural activity, but also as a functional policy network in which adaptive behavior emerges from biologically grounded neural dynamics.

Discussion. Our evaluation follows a progression from MLP and RNN controllers to recurrent LIF-SNNs, fixed homogeneous Izhikevich networks, and EDEN with neuron-specific trainable

Izhikevich dynamics. Hodgkin–Huxley models offer mechanistic ion-channel detail but are expensive for large recurrent policy search, whereas the basic LIF model is efficient but exposes only a limited intrinsic dynamical repertoire [1, 37]. The two-state Izhikevich model provides a middle ground: four intrinsic parameters can generate a broad range of firing responses while retaining a compact numerical formulation [13, 14]. This parameterization makes intrinsic response properties available to optimization. Neuron-specific intrinsic parameters can distribute these responses across multiple time scales, consistent with prior evidence that neuronal heterogeneity can reshape temporal computation and population dynamics [47, 48]. Evolution strategies are technically compatible with this architecture because they can jointly optimize input and output mappings, recurrent weights, and intrinsic parameters from closed-loop episodic returns without differentiating through spike resets, long recurrent trajectories, or environment dynamics; independent parameter perturbations also map naturally to batch evaluation [16, 31].

Limitations and Future Work. The heterogeneous Izhikevich neuron model used in EDEN requires the sequential numerical integration of differential equations. The current implementation lacks a specialized computational architecture optimized for large-scale neurodynamic simulation, such as a neuromorphic chip or FPGA accelerator. This limitation slows training convergence.

EDEN completely dispenses with gradient computation. This avoids the optimization barriers posed by non-differentiable dynamics, but it also prevents EDEN from exploiting a central advantage of modern reinforcement learning (RL). At the policy-optimization level, EDEN can only perform an open-loop "black-box" search based on cumulative episodic return. Unlike PPO or SAC, it does not use temporal-difference errors or value functions for fine-grained credit assignment [16, 42, 43]. Its search efficiency may therefore decrease substantially on sparse-reward tasks or tasks with long-term dependencies, whereas dense learning signals from critic networks can often guide RL methods toward effective regions of the search space more quickly.

EDEN applies a uniform random-initialization scheme and the same ES search mechanism to the neuronal parameters (a, b, c, d) and synaptic weights, which may result in substantial unproductive exploration in a high-dimensional parameter space. Recent work, particularly studies based on the whole-brain connectome of adult *Drosophila*, indicates that measured biological connectivity can provide a strong inductive bias for control [49, 50]. Even when the biological prior is reduced to the directed connectivity topology, without synaptic weights or neurotransmitter types, it can guide a trained controller toward stable rhythmic locomotion [50]. These findings suggest that, for more complex tasks such as visual navigation and multimodal integration, incorporating priors derived from biological data, including the firing properties of specific neuronal populations and known circuit modules, may be more efficient than relying on purely gradient-free search. Initializing EDEN's network topology from the connectomes of model organisms such as *Drosophila* may therefore be an important route toward improving scalability and training efficiency.

Acknowledgments and Disclosure of Funding

Acknowledgements. We would like to thank Dr. Ruxue Wang for her valuable assistance in the preparation of the manuscript and figures.

Funding Sources. This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

References

[1] Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500, 1952.

[2] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943. doi: 10.1007/BF02478259.

[3] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, October 1986. ISSN 1476-4687. doi: 10.1038/323533a0. URL `http://dx.doi.org/10.1038/323533a0`.

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

[5] Anthony M Zador. A critique of pure learning and what artificial neural networks can learn from animal brains. *Nature communications*, 10(1):3770, 2019. doi: 10.1038/s41467-019-11786-6.

[6] C. Van Vreeswijk and H. Sompolinsky. Chaos in neuronal networks with balanced excitatory and inhibitory activity. *Science*, 274, 1996. ISSN 00368075. doi: 10.1126/science.274.5293.1724.

[7] Nicolas Brunel. Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons. *Journal of Computational Neuroscience*, 8, 2000. ISSN 09295313. doi: 10.1023/A:1008925309027.

[8] Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10, 1997. ISSN 08936080. doi: 10.1016/S0893-6080(97)00011-7.

[9] Emre O. Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, 36, 2019. ISSN 15580792. doi: 10.1109/MSP.2019.2931595.

[10] Guillaume Bellec, Darjan Salaj, Anand Subramoney, Robert Legenstein, and Wolfgang Maass. Long short-term memory and learning-to-learn in networks of spiking neurons. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[11] Friedemann Zenke and Surya Ganguli. SuperSpike: Supervised learning in multilayer spiking neural networks. *Neural Computation*, 30(6):1514–1541, 2018. doi: 10.1162/neco_a_01086.

[12] Luca Zanatta, Francesco Barchi, Simone Manoni, Silvia Tolu, Andrea Bartolini, and Andrea Acquaviva. Exploring spiking neural networks for deep reinforcement learning in robotic tasks. *Scientific Reports*, 14(1):30648, 2024. doi: 10.1038/s41598-024-77779-8.

[13] Eugene M. Izhikevich. Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, 14(6):1569–1572, 2003. doi: 10.1109/TNN.2003.820440.

[14] Eugene M. Izhikevich. Which model to use for cortical spiking neurons? *IEEE Transactions on Neural Networks*, 15(5):1063–1070, 2004. doi: 10.1109/TNN.2004.832719.

[15] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. The MIT press, 1992.

[16] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

[17] Peng Qu, Hui Lin, Meng Pang, Xiaofei Liu, Weimin Zheng, and Youhui Zhang. Enlarge: An efficient snn simulation framework on gpu clusters. *IEEE Transactions on Parallel and Distributed Systems*, 34, 2023. ISSN 15582183. doi: 10.1109/TPDS.2023.3291825.

[18] Guangzhi Tang, Neelesh Kumar, Raymond Yoo, and Konstantinos P. Michmizos. Deep reinforcement learning with population-coded spiking neural network for continuous control. In *Proceedings of the Conference on Robot Learning*, pages 2016–2029. PMLR, 2021.

[19] Duzhen Zhang, Tielin Zhang, Shuncheng Jia, and Bo Xu. Multi-sacle dynamic coding improved spiking actor network for reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pages 59–67, 2022.

[20] Guan Wang, Yuhao Sun, Sijie Cheng, and Sen Song. Evolving connectivity for recurrent spiking neural networks. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

[21] Changze Lv, Dongqi Han, Yansen Wang, Xiaoqing Zheng, Xuanjing Huang, and Dongsheng Li. Advancing spiking neural networks for sequential modeling with central pattern generators. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

[22] Zepei Li, Yuhan Shan, and Hongwei Mo. TBC-HRL: A bio-inspired framework for stable and interpretable hierarchical reinforcement learning. *Biomimetics*, 10(11):715, 2025. doi: 10.3390/biomimetics10110715.

[23] Christos Sourmpis, Carl C. H. Petersen, Wulfram Gerstner, and Guillaume Bellec. Trial matching: Capturing variability with data-constrained spiking neural networks. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

[24] Liwei Huang, Zhengyu Ma, Liutao Yu, Huihui Zhou, and Yonghong Tian. Long-range feedback spiking network captures dynamic and static representations of the visual cortex under movie stimuli. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

[25] Matthijs Pals, A. Erdem Sağtekin, Felix Pei, Manuel Gloeckler, and Jakob H. Macke. Inferring stochastic low-rank recurrent neural networks from neural data. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

[26] Jaivardhan Kapoor, Auguste Schulz, Julius Vetter, Felix Pei, Richard Gao, and Jakob H. Macke. Latent diffusion for neural spiking data. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

[27] Sumit Bam Shrestha and Garrick Orchard. SLAYER: Spike layer error reassignment in time. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[28] Mahmoud Akl, Deniz Ergene, Florian Walter, and Alois Knoll. Toward robust and scalable deep spiking reinforcement learning. *Frontiers in Neurorobotics*, 16:1075647, 2023. doi: 10.3389/fnbot.2022.1075647.

[29] Jessica Hunter, Md Maruf Hossain Shuvo, and Krishna Roy. Spiking neural networks for continuous control: Neuromorphic reinforcement learning in conventional computing. ICLR 2026 2nd Workshop on World Models, 2026.

[30] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001. doi: 10.1162/106365601750190398.

[31] Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. Natural evolution strategies. *Journal of Machine Learning Research*, 15(27):949–980, 2014. URL `http://jmlr.org/papers/v15/wierstra14a.html`.

[32] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*, 2017.

[33] Avery A. Sharp, Michael B. O'Neil, L. F. Abbott, and Eve Marder. The dynamic clamp: Artificial conductances in biological neurons. *Trends in Neurosciences*, 16(10):389–394, 1993.

[34] Zachary F. Mainen and Terrence J. Sejnowski. Reliability of spike timing in neocortical neurons. *Science*, 268(5216):1503–1506, 1995.

[35] Emery N. Brown, Robert E. Kass, and Partha P. Mitra. Multiple neural spike train data analysis: State-of-the-art and future challenges. *Nature Neuroscience*, 7(5):456–461, 2004. doi: 10.1038/nn1228.

[36] Robert E. Kass, Valérie Ventura, and Emery N. Brown. Statistical issues in the analysis of neuronal data. *Journal of Neurophysiology*, 94(1):8–25, 2005.

[37] Wulfram Gerstner, Werner M. Kistler, Richard Naud, and Liam Paninski. Neuronal Dynamics, 2014. URL `http://dx.doi.org/10.1017/CBO9781107447615`.

[38] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[39] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IEEE International Conference on Intelligent Robots and Systems*, 2012. doi: 10.1109/IROS.2012.6386109.

[40] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1329–1338. PMLR, 2016. URL `https://proceedings.mlr.press/v48/duan16.html`.

[41] Cecilia Gallego-Carracedo, Matthew G. Perich, Raeed H. Chowdhury, Lee E. Miller, and Juan Álvaro Gallego. Local field potentials reflect cortical population dynamics in a region-specific and frequency-dependent manner. *eLife*, 11:e73155, 2022. doi: 10.7554/eLife.73155. URL `https://doi.org/10.7554/eLife.73155`.

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

[43] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1861–1870. PMLR, 2018.

[44] John P. Cunningham and Byron M. Yu. Dimensionality reduction for large-scale neural recordings. *Nature Neuroscience*, 17(11):1500–1509, 2014. doi: 10.1038/nn.3776.

[45] David Xing, Mehdi Aghagolzadeh, Wilson Truccolo, Erwan Bezard, Gregoire Courtine, and David Borton. Low-dimensional motor cortex dynamics preserve kinematics information during unconstrained locomotion in nonhuman primates. *Frontiers in Neuroscience*, 13:1046, 2019. doi: 10.3389/fnins.2019.01046.

[46] C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax: A differentiable physics engine for large scale rigid body simulation. *arXiv preprint arXiv:2106.13281*, 2021.

[47] Nicolas Perez-Nieves, Vincent C. H. Leung, Pier Luigi Dragotti, and Dan F. M. Goodman. Neural heterogeneity promotes robust learning. *Nature Communications*, 12(1):5791, 2021. doi: 10.1038/s41467-021-26022-3.

[48] Richard Gast, Sara A. Solla, and Ann Kennedy. Neural heterogeneity controls computations in spiking neural networks. *Proceedings of the National Academy of Sciences of the United States of America*, 121(3):e2311885121, 2024. doi: 10.1073/pnas.2311885121.

[49] Sven Dorkenwald, Arie Matsliah, Amy R. Sterling, Philipp Schlegel, Szi-Chieh Yu, Claire E. McKellar, Albert Lin, Marta Costa, Katharina Eichler, Yijie Yin, et al. Neuronal wiring diagram of an adult brain. *Nature*, 634(8032):124–138, 2024. doi: 10.1038/s41586-024-07558-y.

[50] Zehao Jin, Yaoye Zhu, Chen Zhang, and Yanan Sui. Whole-brain connectomic graph model enables whole-body locomotion control in fruit fly. *arXiv preprint arXiv:2602.17997*, 2026.

Appendix

A EDEN

A.1 EDEN Neurodynamic Model

The goal of a neurodynamic model is to describe membrane-potential evolution, spike generation, recovery dynamics, and synaptic propagation within a computationally tractable framework. The most detailed biophysical description is typically given by the Hodgkin–Huxley (HH) model, which explains membrane-potential changes as charge accumulation on a capacitive membrane and explicitly models sodium, potassium, and leak currents:

$$C_m \frac{dv}{dt} = I(t) - g_{\mathrm{Na}} m^3 h (v - E_{\mathrm{Na}}) - g_{\mathrm{K}} n^4 (v - E_{\mathrm{K}}) - g_{\mathrm{L}} (v - E_{\mathrm{L}}), \tag{12}$$

where v is the membrane potential, C_m is the membrane capacitance, $I(t)$ is the external input current, and m, h, and n are ion-channel gating variables. Although the HH model provides an accurate mechanistic description of action-potential generation, it requires integrating multiple nonlinear state variables for each neuron, making it computationally expensive for repeated rollouts of large candidate populations.

To reduce computational complexity, the fast sodium activation variable can be treated as approximately instantaneous, while the slower potassium activation and sodium inactivation processes can be summarized by a single recovery variable. This reduces the ion-channel dynamics to two core states: the membrane potential v_i and the recovery variable u_i. Here, v_i represents the excitability state of the neuron, whereas u_i captures slow negative feedback on membrane-potential evolution. The Izhikevich model follows this principle and represents neuronal dynamics as

$$\frac{dv_i}{dt} = 0.04 v_i^2 + 5 v_i + 140 - u_i + I_i(t), \tag{13}$$

$$\frac{du_i}{dt} = a_i (b_i v_i - u_i). \tag{14}$$

When the membrane potential reaches the firing threshold, the full action-potential waveform is replaced by a reset rule:

$$\text{if } v_i \geq 30\ \mathrm{mV}, \quad v_i \leftarrow c_i, \quad u_i \leftarrow u_i + d_i. \tag{15}$$

This formulation can be viewed as a low-dimensional phenomenological approximation to the HH model. The rising phase of the action potential is represented by a threshold event rather than explicit sodium-channel activation, while repolarization and adaptation after a spike are summarized by u_i, c_i, and d_i. As a result, the model preserves key biological properties such as firing thresholds, membrane recovery, and adaptive spiking, while reducing each neuron to only two dynamical state variables.

In EDEN, the parameters a_i, b_i, c_i, and d_i are not treated as fixed constants shared by all neurons. Instead, they are neuron-specific and trainable intrinsic properties. The parameter a_i controls the recovery time scale, b_i controls the sensitivity of the recovery variable to the membrane potential, c_i determines the reset potential after a spike, and d_i determines the post-spike recovery increment. This design enables EDEN to contain diverse neuronal dynamics within a single network, including fast-spiking, adaptive-spiking, and strongly recovering neurons, thereby improving dynamical expressiveness while preserving low computational cost.

The total input current to neuron i is defined as

$$I_i(t) = I_i^{\mathrm{obs}}(t) + I_i^{\mathrm{syn}}(t) + \xi_i(t), \tag{16}$$

where $I_i^{\mathrm{obs}}(t)$ is the observation-driven input, $I_i^{\mathrm{syn}}(t)$ is the recurrent synaptic input, and $\xi_i(t)$ is a stochastic perturbation term. The observation input is obtained through a linear mapping from the environment state:

$$I^{\mathrm{obs}}(t) = W_{\mathrm{in}} o_t + b_{\mathrm{in}}. \tag{17}$$

The stochastic perturbation follows $\xi_i(t) \sim \mathcal{N}(0, \sigma_\xi^2)$. This term captures intrinsic neural variability and also introduces additional exploration during policy search.

A.2 EDEN Synapse Model

Biological synaptic transmission involves the arrival of presynaptic action potentials, neurotransmitter release, postsynaptic receptor activation, and ion flow that changes the postsynaptic membrane potential. A biophysically detailed synaptic current is commonly written in conductance-based form:

$$I_i^{\mathrm{syn}}(t) = \sum_j g_{ij}(t)\left(E_{ij} - v_i(t)\right), \tag{18}$$

where $g_{ij}(t)$ is the synaptic conductance from presynaptic neuron j to postsynaptic neuron i, E_{ij} is the reversal potential, and $v_i(t)$ is the postsynaptic membrane potential. The reversal potential determines whether a synapse is excitatory or inhibitory: synapses with reversal potentials above the membrane potential tend to depolarize the postsynaptic neuron, whereas those below it tend to hyperpolarize the neuron.

To reduce computational cost, the receptor-channel opening process can be abstracted as a temporal response kernel triggered by presynaptic spikes. If neuron j fires at time $t_j^{(f)}$, the synaptic conductance can be written as

$$g_{ij}(t) = \bar{g}_{ij} \sum_f s\left(t - t_j^{(f)}\right), \tag{19}$$

where $\bar{g}_{ij}$ denotes synaptic strength and $s(\cdot)$ is the synaptic response kernel. A common exponential response is

$$s(t) = \exp(-t/\tau_{\mathrm{syn}})H(t), \tag{20}$$

where τ_{syn} is the synaptic time constant and $H(t)$ is the Heaviside step function. Although this formulation retains a biological interpretation of decaying synaptic conductance, it requires maintaining and updating synaptic states at every simulation step, which is costly when evaluating many candidate networks in parallel.

EDEN therefore adopts a simplified event-driven synapse approximation. The model preserves directed synaptic weights and excitatory/inhibitory signs, but does not explicitly integrate conductance decay. When a presynaptic spike occurs, it directly injects an instantaneous current into postsynaptic neurons according to the corresponding synaptic weight:

$$I_i^{\mathrm{syn}}[n] = \sum_j w_{ij} z_j[n], \tag{21}$$

where $z_j[n] \in \{0, 1\}$ indicates whether neuron j fires at discrete time step n, and w_{ij} is the synaptic weight from neuron j to neuron i. This is equivalent to approximating the synaptic response kernel by an impulse function, $s[n] = \delta[n]$, so that

$$I_i^{\mathrm{syn}}[n] = \sum_j w_{ij} \sum_f \delta[n - n_j^{(f)}]. \tag{22}$$

Under this approximation, the distinction between conductance $g_{ij}(t)$ and reversal potential E_{ij} is absorbed into the signed weight w_{ij}: positive weights represent excitatory inputs, while negative weights represent inhibitory inputs. This preserves the biological structure that presynaptic spikes influence postsynaptic neurons through directed connections, while avoiding continuous state integration for every synapse.

Computationally, this design turns synaptic propagation into a sparse event-accumulation process. Only outgoing connections of neurons that fire at the current time step are accessed, whereas connections from silent neurons do not contribute to the current update. Therefore, the cost of synaptic propagation depends mainly on the number of spike events and their outgoing connections, rather than the total number of synapses in the network. By compressing synaptic transmission into signed connection weights and event-triggered accumulation, EDEN maintains circuit-level interpretability while improving the efficiency of large-scale candidate rollout evaluation.

A.3 Network Parameters

The main network parameters used in EDEN are summarized in Table 2.For initial individuals, intrinsic neuronal parameters are sampled from biologically plausible ranges: $a_i \in [0.02, 0.10]$,

Table 2: Neuron and synapse parameters used in EDEN.

Parameter	Value / Range	Unit
Firing threshold	30	mV
Recovery time scale a_i	$[0.02, 0.10]$	–
Recovery sensitivity b_i	$[0.20, 0.25]$	–
Reset potential c_i	$[-65.0, -50.0]$	mV
Post-spike recovery strength d_i	$[2.0, 8.0]$	–
Synaptic weight w_{ij}	trainable / searchable	–
Noise scale σ_ξ	1.0	–

$b_i \in [0.20, 0.25]$, $c_i \in [-65, -50]$, and $d_i \in [2, 8]$. Recurrent synaptic weights are initialized with scale normalization according to the number of neurons, while input and output mapping parameters follow standard neural-network initialization schemes.

At each environment time step, EDEN performs an internal neurodynamic simulation before action generation. Specifically, each environment interaction step corresponds to 200 internal neuronal simulation steps with a step size of 0.1 ms. Thus, the neurodynamic core evolves for 20 ms before each action is produced, which is on the same order as the environment sampling interval in MuJoCo continuous-control tasks.

B Artificial Neural Networks

In addition to the proposed biologically grounded neurodynamic model EDEN, we consider artificial neural-network baselines to separate the contribution of heterogeneous neural dynamics from that of generic nonlinear function approximation and recurrence.

B.1 Multilayer Perceptron

A multilayer perceptron (MLP) is a feedforward neural network that computes deterministic continuous transformations through stacked affine mappings and pointwise nonlinearities. Given an observation o_t, the MLP policy is defined as

$$h_t^{(1)} = \phi(W_1 o_t + b_1), \qquad h_t^{(2)} = \phi(W_2 h_t^{(1)} + b_2), \tag{23}$$

$$a_t = a_{\max} \tanh(W_a h_t^{(2)} + b_a). \tag{24}$$

The MLP operates as a static observation-to-action mapping. It does not maintain an explicit recurrent state and does not contain biologically interpretable neural dynamics. Its effectiveness mainly arises from architectural simplicity, differentiability, and compatibility with efficient gradient-based training. In this work, the MLP is used as an artificial neural-network baseline trained under the same evolutionary optimization setting.

B.2 Recurrent Neural Network

A recurrent neural network (RNN) is a standard artificial model for sequential and temporally structured data. Unlike feedforward architectures, an RNN maintains an internal hidden state through recurrent connections:

$$h_{t+1} = \phi(W_o o_t + W_h h_t + b_h), \qquad a_t = a_{\max} \tanh(W_a h_{t+1} + b_a). \tag{25}$$

Although the RNN introduces temporal recurrence, its recurrent activity is represented by abstract hidden states rather than explicit membrane-potential dynamics, spike-triggered resets, or neuron-specific intrinsic parameters. The RNN therefore serves as an important baseline for distinguishing generic recurrent computation from biologically grounded neurodynamic recurrence. The key differences among MLP, RNN, SNN, and EDEN are summarized in Table 3.

Table 3: Comparison of different policy network architectures.

Property	MLP	RNN	SNN	EDEN
Basic unit	Weighted sum + activation	Recurrent artificial unit	Fixed Izhikevich neuron	Heterogeneous Izhikevich neuron
State dynamics	Static	Abstract hidden-state recurrence	Membrane potential + recovery variable	Heterogeneous membrane potential + recovery variable
Information coding	Continuous activations	Continuous hidden state	Spike events + neural states	Spike events + heterogeneous neural states
Intrinsic parameters	None	None	Fixed shared (a, b, c, d)	Neuron-specific trainable (a_i, b_i, c_i, d_i)
Synaptic model	Dense affine weights	Dense recurrent affine weights	Event-driven recurrent spike synapses with trainable signed weights	Event-driven recurrent spike synapses with trainable signed weights
Biological interpretability	Low	Low	Moderate	High

C Parallel Simulation Framework

The parallel simulation backend of EDEN is built upon the high-performance spiking neural-network simulator ENLARGE [17], enabling large-scale evaluation of biological neurodynamic models during evolutionary optimization. ENLARGE is designed for event-driven computation in spiking neural networks and provides fine-grained network representation, sparse spike propagation, and parallel execution mechanisms. These features allow EDEN to efficiently evaluate population-level candidate rollouts while preserving the biological interpretability of neuronal and synaptic dynamics.

EDEN inherits and extends the fine-grained representation in ENLARGE by organizing neuronal states, intrinsic neuronal parameters, synaptic weights, and network topology as explicit simulation objects. Membrane potentials, recovery variables, and neuron-specific parameters a_i, b_i, c_i, and d_i are stored in structured arrays, while synaptic weights w_{ij} and connectivity patterns are maintained separately. This representation provides a direct parameter interface for evolutionary optimization, allowing neuronal dynamics, synaptic strengths, and circuit structure to be encoded, perturbed, and selected as searchable components of each candidate network.

During simulation, EDEN adopts a spike-list-driven event propagation mechanism. At each simulation step, neuronal membrane potentials and recovery variables are updated in parallel, and spike events are generated according to the threshold condition:

$$z_i[n] = H(v_i[n] - v_{\mathrm{th}}). \tag{26}$$

The indices of spiking neurons are then compressed into a spike list $\mathcal{F}[n]$, which triggers synaptic propagation. For the event-driven recurrent spike synapses used in EDEN, the recurrent input current is computed as

$$I_i^{\mathrm{syn}}[n] = \sum_{j \in \mathcal{F}[n]} w_{ij}. \tag{27}$$

This mechanism restricts synaptic computation to neurons that actually fire and their outgoing connections, making the computational cost of each simulation step adapt to the number of spike events. Since biologically plausible spiking networks typically exhibit sparse firing activity, event-driven propagation substantially improves rollout efficiency.

EDEN further uses sparse topology representations to organize neuronal connectivity. The network topology is stored in a compressed sparse-row-like format, where each presynaptic neuron is associated only with its existing outgoing connections and corresponding synaptic parameter indices. This design allows the evolutionary algorithm to search over sparse neural circuits while ensuring that only valid connections are accessed during simulation. Structural sparsity in neural circuits and temporal sparsity in spike activity jointly reduce the cost of synaptic propagation.

During population evaluation, EDEN exploits two levels of parallelism. The first level is within-candidate parallelism, where neuronal state updates and synaptic event propagation are parallelized inside a single candidate network. The second level is population-level parallelism, where multiple candidate networks or rollouts are evaluated concurrently during evolutionary optimization. The former reduces the time required for a single neurodynamic simulation, whereas the latter improves the throughput of fitness evaluation at each generation. Through this computational organization, EDEN can evaluate large populations of candidate networks with heterogeneous neuronal parameters and trainable synaptic structures.

Overall, EDEN tightly integrates ENLARGE's high-performance SNN simulation mechanisms with evolutionary optimization. Fine-grained representation provides candidate networks with explicit biophysical parameterization, sparse event propagation reduces the cost of neurodynamic simulation, and parallel execution improves population-level evaluation throughput. This framework enables EDEN to support both biologically interpretable modelling and efficient policy search.

D Parallel Evolutionary Algorithm

This work uses OpenAI-ES to optimize EDEN as a black-box policy [16]. The algorithm follows a parallel rollout-and-update procedure: the master process samples candidate EDEN parameter sets around the current parameter center, worker nodes evaluate these candidates through closed-loop rollouts in MuJoCo, and the master process updates the parameter center using the returned fitness values. The implementation retains the Gaussian-perturbation and utility-weighted ES formulation introduced in Section 3.3, and realizes it with antithetic candidate pairs, centered-rank fitness shaping, unperturbed center replicas, fixed parameter-wise perturbation multipliers and training masks, and an AdamW center update.

D.1 Gaussian-smoothed objective and the basic ES estimator

Let $F(\theta)$ denote the cumulative return of a policy with parameters θ over a complete closed-loop episode. The Gaussian-smoothed objective around the current parameter center m_g is

$$\widetilde{F}(m_g) = \mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)}\left[F(m_g + \sigma\epsilon)\right]. \tag{28}$$

Using the log-likelihood identity, its gradient with respect to the parameter center can be written as

$$\nabla_{m_g}\widetilde{F}(m_g) = \frac{1}{\sigma}\mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)}\left[F(m_g + \sigma\epsilon)\epsilon\right]. \tag{29}$$

A Monte Carlo approximation with λ Gaussian samples gives the raw-return estimator

$$\widehat{g}_{F,g} = \frac{1}{\lambda\sigma}\sum_{k=1}^{\lambda} F(\theta_k)\epsilon_k. \tag{30}$$

The practical ES update replaces raw returns with fitness-shaped utilities u_k, defining the search direction

$$\widehat{g}_g = \frac{1}{\lambda\sigma}\sum_{k=1}^{\lambda} u_k\epsilon_k, \qquad m_{g+1} = m_g + \eta\widehat{g}_g. \tag{31}$$

Equation (31) is the update stated in the Methods section. The distinction between the two estimators is important: $\widehat{g}_{F,g}$ is a Monte Carlo gradient estimator for the Gaussian-smoothed return objective, whereas $\widehat{g}_g$ is the search direction after fitness shaping. OpenAI-ES augments this basic update with antithetic sampling and adaptive optimization of the parameter center [16].

D.2 Antithetic sampling

Assume that λ is even and define the number of perturbation pairs as $M = \lambda/2$. For pair i, sample

$$\epsilon_i \sim \mathcal{N}(0, I), \qquad i = 1, \ldots, M, \tag{32}$$

and construct positive and negative candidates:

$$\theta_i^p = m_g + \sigma\epsilon_i, \qquad \theta_i^n = m_g - \sigma\epsilon_i. \tag{33}$$

The superscripts p and n denote positive and negative perturbations. Let F_i^p and F_i^n be their complete-episode returns, and let u_i^p and u_i^n be the corresponding utilities. Because the negative candidate uses noise $-\epsilon_i$, the sum over the $\lambda = 2M$ candidates in Eq. (31) becomes

$$\widehat{g}_g = \frac{1}{2M\sigma} \sum_{i=1}^{M} (u_i^p - u_i^n)\, \epsilon_i. \tag{34}$$

Antithetic sampling therefore retains the basic search direction in the Methods section while estimating the effect of each sampled direction from the positive-negative utility difference.

D.3 Centered-rank fitness shaping

The implementation does not use raw return magnitudes directly. Instead, it ranks the $\lambda = 2M$ search candidates jointly. The centered-rank utility of search candidate k is

$$u_k = \frac{\mathrm{rank}(F_k)}{\lambda - 1} - \frac{1}{2}, \qquad u_k \in [-0.5, 0.5]. \tag{35}$$

The worst return has rank 0 and the best return has rank $\lambda - 1$. This transformation preserves the ordering of candidates while reducing the effect of task-specific reward scales and outlying return magnitudes on the update. The utilities of the positive and negative candidates, u_i^p and u_i^n, are then substituted into Eq. (34).

D.4 Applied perturbations and center replicas

The implementation permits fixed perturbation multipliers for different parameter groups and allows selected parameters to remain frozen. Let D denote the fixed elementwise perturbation multiplier and training mask. The applied perturbation is

$$\delta_i = \sigma D \epsilon_i, \qquad \theta_i^p = m_g + \delta_i, \qquad \theta_i^n = m_g - \delta_i. \tag{36}$$

When all trainable parameters share the same perturbation scale, $D = I$, and the candidates reduce to the form $m_g + \sigma\epsilon_k$ used in the Methods section. Entries of D corresponding to frozen parameters are zero.

In addition to the $\lambda = 2M$ search candidates, each generation includes E unperturbed replicas of the current parameter center. Thus, the total number of policies evaluated in parallel is $\lambda + E$. Their mean return is

$$F_{\mathrm{center}}(m_g) = \frac{1}{E} \sum_{j=1}^{E} F(m_g; \xi_j), \tag{37}$$

where ξ_j denotes the environment initialization or evaluation trajectory of replica j. Center replicas are used only to monitor the online performance of the current parameter center and to nominate candidate checkpoints. They are excluded from both the ranking of the λ search candidates and the search-direction calculation.

D.5 AdamW update of the parameter center

The implementation constructs the ascent direction directly from the applied perturbations δ_i:

$$d_g = \frac{1}{M} \sum_{i=1}^{M} (u_i^p - u_i^n)\, \delta_i. \tag{38}$$

For fixed σ and $D = I$, $d_g = 2\sigma^2 \widehat{g}_g$. Thus, d_g has the same ascent direction as the OpenAI-ES estimate in Eq. (34), up to a positive global scale factor. The implementation absorbs this numerical scale through the AdamW learning rate and adaptive normalization rather than treating d_g as a normalized, unbiased gradient estimator. Because AdamW follows a descent direction, the optimizer receives $-d_g$:

$$(m_{g+1}, s_{g+1}) = \mathrm{AdamWStep}\left(m_g, s_g, -d_g; \eta, \lambda_{\mathrm{wd}}\right), \tag{39}$$

where s_g is the AdamW optimizer state, η retains the learning-rate notation from the Methods section, and λ_{wd} is the weight decay coefficient. The implementation keeps σ and D fixed; it neither learns a search covariance nor explicitly computes a Fisher natural gradient.

D.6 Parallel evaluation and model selection

Following the workflow in the Methods section, every search candidate executes a complete closed-loop episode in MuJoCo. Candidate fitness is the cumulative return. If a candidate completes multiple full episodes within a generation, its fitness is the mean return over those episodes. Once all search candidates have been evaluated, the master node computes centered-rank utilities jointly and updates the parameter center.

The center-replica score $F_{\mathrm{center}}(m_g)$ is used to monitor training and nominate candidate checkpoints. After training, nominated parameter centers are evaluated in independent episodes. These final evaluation results are reported separately from the online center-replica returns observed during training.

D.7 Algorithm summary

The parallel execution procedure follows the same rollout-and-update structure, with OpenAI-ES determining how candidates are generated and how the parameter center is updated.

Algorithm 1: Parallel Evolutionary Optimization for EDEN

Input: Initial EDEN parameter center m_0, search population size λ, center replicas E, perturbation scale σ, fixed multiplier and mask D, learning rate η, and weight decay λ_{wd}.
Output: Best network parameters θ_{best}.
Initialise: Parameter center m_0, AdamW state s_0, and worker pool.
while *generation < generation limit* **do**
  Sample $M = \lambda/2$ perturbations $\delta_i = \sigma D \epsilon_i$ and form antithetic candidates $\{\theta_i^p, \theta_i^n\}_{i=1}^M$; append E center replicas.
  Dispatch all $\lambda + E$ candidates to worker nodes for parallel rollout evaluation in MuJoCo.
  Compute cumulative episode return for each candidate.
  Return all fitness values to the master process.
  Rank the λ search returns to obtain centered-rank utilities $\{u_i^p, u_i^n\}_{i=1}^M$.
  Compute d_g from the paired utility differences and update m_g using AdamW.
  Record the evaluated center as θ_{best} when its mean center-replica return improves.
end
return θ_{best} *for independent final evaluation*

E Reinforcement Learning Reference Algorithms

Two representative deep reinforcement learning algorithms are considered, PPO [42] and SAC [43], as reference performance levels for MuJoCo continuous-control tasks. These methods are not strictly paradigm-matched baselines for EDEN, because they rely on gradient-based optimization of conventional MLP policies, whereas EDEN optimizes heterogeneous neurodynamic networks through evolutionary search. Therefore, PPO and SAC are used to contextualize the achieved control performance rather than to provide a direct algorithmic comparison.

PPO is an on-policy actor–critic algorithm based on a clipped surrogate objective, while SAC is an off-policy maximum-entropy actor–critic algorithm that learns from a replay buffer and uses entropy regularization for exploration. Accordingly, the two methods require different training hyperparameters. The reference hyperparameters used for PPO and SAC are listed in Tables 4 and 5, respectively.

F Experiments

F.1 MuJoCo Tasks

The MuJoCo tasks used in this work are standard robotic continuous-control benchmarks from OpenAI Gym and MuJoCo [38, 39]. These environments involve high-dimensional continuous state spaces and continuous action spaces, where observations typically include positions, velocities,

Table 4: PPO hyperparameters used for MuJoCo reference experiments.

Parameter	**Value**	**Description**
Policy network	MLP	Two hidden layers with 64 units each
Activation function	Tanh	Hidden-layer nonlinearity
Learning rate	3×10^{-4}	Adam optimizer
Discount factor (γ)	0.99	Reward discounting
GAE lambda (λ)	0.95	Generalized Advantage Estimation
Rollout length	2048	Number of environment steps per update
Mini-batch size	64	Mini-batch size for policy/value updates
Epochs per update	10	Number of optimization epochs per rollout
Clipping range	0.2	PPO clipped objective coefficient
Entropy coefficient	0.0	Entropy regularization weight
Value-function coefficient	0.5	Weight of value-function loss
Max gradient norm	0.5	Gradient clipping threshold

Table 5: SAC hyperparameters used for MuJoCo reference experiments.

Parameter	**Value**	**Description**
Policy network	MLP	Two hidden layers with 64 units each
Critic network	MLP	Two Q-networks with 64 hidden units per layer
Activation function	ReLU	Hidden-layer nonlinearity
Learning rate	3×10^{-4}	Adam optimizer for actor and critics
Discount factor (γ)	0.99	Reward discounting
Replay buffer size	10^6	Maximum number of stored transitions
Mini-batch size	256	Mini-batch size for gradient updates
Learning starts	10,000	Initial random interaction steps before updates
Soft update coefficient (τ)	0.005	Target-network update coefficient
Train frequency	1	Update frequency in environment steps
Gradient steps	1	Number of gradient updates per environment step
Entropy coefficient	Auto	Automatically tuned temperature parameter
Target entropy	$-\dim(\mathcal{A})$	Target policy entropy for automatic tuning

joint angles, and angular velocities of robot body parts, and actions correspond to torque or force commands applied to actuated joints. The reward functions mainly encourage task-specific forward movement or balance, with additional penalties on control energy, instability, or unhealthy states. Following the standard benchmark setting, each episode is truncated at a maximum horizon of 1000 environment steps.

The temporal resolution of each environment is determined by the MuJoCo simulator timestep and the environment-specific frame-skip parameter. Specifically, one environment interaction step corresponds to

$$\Delta t_{\mathrm{env}} = \mathrm{frame_skip} \times \Delta t_{\mathrm{sim}}, \tag{40}$$

where Δt_{sim} is the internal MuJoCo simulation timestep. This is important for EDEN because each policy action is generated after a short internal neurodynamic simulation. In our implementation, EDEN evolves for 200 internal neuronal simulation steps with a step size of 0.1 ms before producing each action, corresponding to 20 ms of neural dynamics. This timescale is comparable to the control intervals of the MuJoCo tasks, allowing EDEN to operate as a continuous-time neurodynamic controller within closed-loop physical interaction.

These tasks cover a broad spectrum of control difficulty. InvertedDoublePendulum mainly tests balance and stabilization under low-dimensional actuation. Swimmer and HalfCheetah emphasize rhythmic locomotion in relatively low-dimensional systems. Hopper and Walker2d require stable contact-rich locomotion with strict balance constraints. Ant and Humanoid further increase the dimensionality of both state and action spaces, making them suitable for evaluating whether EDEN can scale from simple balance control to complex multi-joint locomotion.

Table 6: Summary of MuJoCo continuous-control tasks used in this work. The environment interval is computed as $\Delta t_{\text{env}} = \text{frame_skip} \times \Delta t_{\text{sim}}$.

Task	**Obs. dim.**	**Act. dim.**	Δt_{env} **(s)**	**Control objective**
InvertedDoublePendulum	11	1	0.05	Balance a two-link pendulum by applying force to the cart
Swimmer	8	2	0.04	Coordinate body joints to swim forward with low control cost
Ant	27	8	0.05	Control a quadruped robot to move forward while remaining healthy
HalfCheetah	17	6	0.05	Drive a planar cheetah robot to run forward efficiently
Hopper	11	3	0.008	Control a one-legged robot to hop forward without falling
Walker2d	17	6	0.008	Coordinate a two-legged walker to move forward stably
Humanoid	376	17	0.015	Control a high-dimensional humanoid to walk forward while maintaining balance

F.2 Interface Between EDEN and MuJoCo

At each MuJoCo control step t, the environment provides a continuous observation vector $o_t \in \mathbb{R}^{d_o}$, which contains task-dependent physical variables such as joint positions, velocities, angles, and angular velocities. Before being injected into the neurodynamic core, the observation is first normalized using running statistics:

$$\hat{o}_t = \frac{o_t - \mu_o}{\sigma_o + \epsilon}, \tag{41}$$

where μ_o and σ_o denote the running mean and standard deviation of observations, and ϵ is a small constant for numerical stability. The normalized observation is then linearly projected into input currents:

$$I_t^{\text{obs}} = \gamma_1 (W_{\text{in}} \hat{o}_t + b_{\text{in}}), \tag{42}$$

where $W_{\text{in}} \in \mathbb{R}^{N \times d_o}$ maps the observation space to the N-neuron population, $b_{\text{in}} \in \mathbb{R}^N$ is the input bias, and γ_1 controls the scale of observation-driven current. During one MuJoCo control interval, this input current is held constant while the EDEN neurodynamic core evolves for K internal simulation steps:

$$(v[n], u[n], z[n]) = \text{EDENStep}(v[n-1], u[n-1], I_t^{\text{obs}}, I^{\text{syn}}[n]), \qquad n = 1, \ldots, K. \tag{43}$$

After the internal simulation, the neural activity over the control interval is converted into a continuous action. Specifically, we aggregate the spike activity of each neuron over the K internal steps:

$$r_t = \frac{1}{K} \sum_{n=1}^{K} z[n], \tag{44}$$

where $z[n] \in \{0, 1\}^N$ is the spike vector at internal step n, and $r_t \in \mathbb{R}^N$ is the population firing-rate vector. The action sent to MuJoCo is decoded from this population activity by

$$a_t = a_{\max} \tanh\left(\gamma_2 W_{\text{out}} r_t + b_{\text{out}}\right), \tag{45}$$

where $W_{\text{out}} \in \mathbb{R}^{d_a \times N}$ maps neural activity to the d_a-dimensional action space, $b_{\text{out}} \in \mathbb{R}^{d_a}$ is the output bias, γ_2 controls the output scale, and $a_{\max}$ denotes the action bound of the MuJoCo environment. The hyperbolic tangent ensures that the decoded action remains within the valid continuous control range.

The resulting action a_t is applied to the MuJoCo simulator for one environment step. The simulator then returns the next observation o_{t+1}, reward r_t^{env}, and termination signal. The cumulative episode return is used as the fitness value for the corresponding EDEN parameter set. In this way, EDEN forms a closed-loop neurodynamic controller: physical states are encoded as input currents, recurrent spiking dynamics transform these currents over biological timescales, and population firing activity is decoded into continuous motor commands.

F.3 Hyperparameter optimization

We conducted multiple hyperparameter searches over the ranges summarized in Table 7. Based on the results across tasks, we recommend the following configuration as a common starting point: 64 neurons, a 10-ms neural simulation window, a 0.1-ms integration step, an input gain of 4, a recurrent gain of 1, an output gain of 5, an ES learning rate of $\eta = 0.1$, and a perturbation scale of $\sigma = 0.15$. This configuration provides a unified initialization across tasks and can be adjusted within a local neighborhood for a specific task.

Table 7: Hyperparameter search ranges for EDEN.

Hyperparameter	Search range
Number of neurons	$\{32, 64, 128, 256\}$
Neural simulation window	$\{10, 20, 30\}$ ms
Integration step	$\{0.1, 0.2, 0.5\}$ ms
Population size	$\{512, 1024, 2048\}$
ES learning rate η	0.003–0.18
ES perturbation scale σ	0.01–0.35

F.4 Wall-clock time and environment-interaction cost

To reduce EDEN's wall-clock training time, we migrated the implementation to the JAX-based MJX runtime. MJX permits many MuJoCo environments to be evaluated in parallel on accelerators such as GPUs, thereby reducing the cost of evaluating population members sequentially and making it better suited to population-based evolutionary optimization.

We measured the wall-clock training time of PPO and SAC at 10^7 and 10^8 environment interactions. The PPO and SAC entries in Table 8 are means over three random seeds. Because episode lengths and early-termination events vary across EDEN runs, their exact interaction counts are not identical. We therefore report one representative EDEN run for each task. The columns should be interpreted as cost measurements at their respective interaction budgets rather than as an equal-budget speed comparison.

Table 8: Wall-clock training time and environment-interaction cost. PPO and SAC times are three-seed means; each EDEN result is one representative run.

	PPO time (s)		SAC time (s)		EDEN	
Task	10^7	10^8	10^7	10^8	Interactions	Time (s)
Ant	121.6	442.4	207.9	797.7	1.00×10^9	3442.4
HalfCheetah	368.4	2525.0	721.9	3833.8	1.00×10^9	17415.2
Hopper	370.9	2191.3	462.5	3174.1	9.87×10^8	12392.8
Humanoid	202.7	771.7	302.4	1213.8	6.58×10^8	11109.3
Walker2d	365.7	2560.2	444.5	2826.1	9.99×10^8	19126.7

MJX parallelization reduces the practical runtime of EDEN, but the current implementation still requires more environment interactions and, on most of these tasks, more wall-clock time than PPO or SAC at the reported budgets. One contributing factor is that the heterogeneous neurodynamic network performs multiple state updates with strong temporal dependencies within every environment step. General-purpose CPU and GPU systems are primarily optimized for matrix operations and highly parallel deep-learning workloads rather than this form of neurodynamic simulation. Future work will investigate parallel computing architectures tailored to EDEN, including neuromorphic hardware, dedicated accelerators, and other hardware implementations for neurodynamic simulation.

F.5 Scalability of EDEN

The 64-neuron configuration used in the main experiments is a common benchmark chosen to balance policy capacity, neurodynamic simulation cost, and the evolutionary search budget. It is not a theoretical upper bound on either EDEN or ES optimization. The approximately 4,350 parameters

discussed in the main text refer to the hidden-core parameters, whereas the actual EDEN search space also includes the remaining optimized policy parameters and therefore already exceeds this value.

To examine the effect of network size directly, we compared 32, 64, 128, and 256 neurons on HalfCheetah, Hopper, Walker2d, and Humanoid using multiple random seeds. Table 9 shows that the effect of network size is task dependent. Simpler tasks are generally easier to optimize with smaller networks, whereas complex, high-dimensional tasks can benefit from larger neural populations. Although Hopper has low observation and action dimensions, early termination after a fall and the associated stability constraints make the search more difficult. Task complexity therefore cannot be characterized by input and output dimensionality alone.

Table 9: Return across EDEN network sizes. Values are mean $\pm$ standard deviation over multiple random seeds.

Task	Obs., act. dim.	32 neurons	64 neurons	128 neurons	256 neurons
HalfCheetah	17, 6	5638 ± 664	5856 ± 836	4808 ± 1326	4960 ± 1895
Hopper	11, 3	1776 ± 853	2674 ± 1249	2651 ± 1329	2427 ± 1177
Walker2d	17, 6	2804 ± 1340	2707 ± 1005	3022 ± 966	2050 ± 435
Humanoid	224, 17	2411 ± 2118	2268 ± 1425	3307 ± 1966	3256 ± 2240

Considering performance stability across tasks, neurodynamic simulation cost, the dimensionality of the evolutionary search, and the need for a common cross-task comparison, we selected 64 neurons as the unified initial setting for the main experiments.

G Full Trainable Parameter Comparison

To further examine whether the performance difference arises from the overall trainable parameter budget, we compare the full number of trainable parameters across EDEN, SNN-ES, MLP-PPO, and MLP-SAC. Unlike the hidden-core comparison, this analysis includes all input, hidden, recurrent, readout, and output-layer parameters. Let d_s denote the observation dimension and d_a denote the action dimension of each MuJoCo task. All models use a 64-unit hidden core or hidden layer.

For EDEN, the full neurodynamic policy consists of input synapses from the observation to the 64-neuron recurrent population, recurrent synapses within the neurodynamic core, readout weights from the recurrent population to the action space, readout biases, and heterogeneous intrinsic Izhikevich parameters. Specifically, the input mapping contains $d_s \times 64$ trainable synaptic weights, the recurrent core contains 64×64 trainable synaptic weights, and the readout contains $64 \times d_a$ weights and d_a biases. In addition, EDEN assigns each neuron an independent set of four trainable intrinsic parameters (a_i, b_i, c_i, d_i), resulting in 4×64 additional parameters. Therefore, the total number of trainable parameters in EDEN is

$$d_s \times 64 + 64 \times 64 + 64 \times d_a + d_a + 4 \times 64 = 64d_s + 65d_a + 4352.$$

For SNN-ES, the overall architecture follows the same input, recurrent, and readout structure as EDEN, but the intrinsic Izhikevich parameters are fixed and shared across neurons rather than being trainable and heterogeneous. Thus, SNN-ES contains $d_s \times 64$ input synapses, 64×64 recurrent synaptic weights, $64 \times d_a$ readout weights, and d_a readout biases. Since the shared neuronal parameters are not optimized, the total number of trainable parameters is

$$d_s \times 64 + 64 \times 64 + 64 \times d_a + d_a = 64d_s + 65d_a + 4096.$$

For MLP-PPO, we count the actor and critic networks separately, following the standard actor–critic formulation used in continuous-control settings. The actor is a two-layer MLP policy with a 64-unit hidden layer, an action mean head, and a trainable log-standard-deviation vector. The actor therefore contains an input layer with $d_s \times 64$ weights and 64 biases, a hidden layer with 64×64 weights and 64 biases, an action mean head with $64 \times d_a$ weights and d_a biases, and a trainable log-standard-deviation vector of size d_a. This gives the actor parameter count

$$64d_s + 66d_a + 4224.$$

The PPO critic is a separate value-function network with the same 64-unit hidden structure and a scalar value head. It contains an input layer with $d_s \times 64$ weights and 64 biases, a hidden layer with 64×64 weights and 64 biases, and a value head with 64×1 weights and one bias, resulting in

$$64d_s + 4289$$

trainable parameters. Therefore, the full MLP-PPO actor–critic model contains

$$(64d_s + 66d_a + 4224) + (64d_s + 4289) = 128d_s + 66d_a + 8513$$

trainable parameters.

For MLP-SAC, the actor is also a stochastic policy network, but it outputs both the action mean and the log-standard deviation as state-dependent quantities. Therefore, the SAC actor contains an input layer, a hidden layer, an action mean head, and a log-standard-deviation head. This results in

$$64d_s + 130d_a + 4224$$

trainable actor parameters. In addition, SAC uses two separate Q-function critics. Each Q critic takes the concatenated state-action vector (s, a) as input, maps it to a 64-unit hidden layer, applies another 64-unit hidden transformation, and outputs a scalar Q value. Thus, each Q critic contains

$$64d_s + 64d_a + 4289$$

trainable parameters. Including two Q critics and one learnable entropy-temperature parameter, the total trainable parameter count of MLP-SAC is

$$(64d_s + 130d_a + 4224) + 2(64d_s + 64d_a + 4289) + 1 = 192d_s + 258d_a + 12803.$$

The target Q networks used in SAC are auxiliary copies for target estimation and are therefore not counted as independently trainable parameters.

As summarized in Table 10, EDEN and SNN-ES optimize only the policy itself, whereas MLP-PPO and MLP-SAC introduce additional critic networks. PPO adds a separate value-function critic for advantage estimation, while SAC further introduces two Q-function critics and a learnable entropy-temperature parameter. Therefore, even when the policy hidden size is fixed to 64 units, actor–critic baselines involve a larger full trainable parameter budget than ES-based policy optimization methods. This comparison highlights that EDEN does not rely on additional critic networks or auxiliary value estimators; instead, its additional parameters are concentrated in the heterogeneous intrinsic dynamics of the neurodynamic policy itself.

Table 10: Full trainable parameter comparison across EDEN, SNN-ES, MLP-PPO, and MLP-SAC.

Algorithm	**Scope**	**Parameter components**	**Count**
EDEN	Full neurodynamic policy	Input synapses, recurrent synapses, readout weights and bias, and heterogeneous Izhikevich parameters.	$64d_s + 65d_a + 4352$
SNN-ES	Full homogeneous spiking policy	Input synapses, recurrent synapses, and readout weights and bias. Shared Izhikevich parameters are fixed and therefore not counted as trainable parameters.	$64d_s + 65d_a + 4096$
MLP-PPO	Full actor–critic model	Actor network with input layer, hidden layer, action mean head, and trainable log-standard-deviation vector; plus a separate value-function critic.	$128d_s + 66d_a + 8513$
MLP-SAC	Full actor–critic model	Actor network with input layer, hidden layer, action mean head, and log-standard-deviation head; plus two separate Q-function critics and one learnable entropy-temperature parameter.	$192d_s + 258d_a + 12803$

H Differences in neuronal connections before and after training

Figure 5 shows the distribution characteristics of synaptic weight probability density before and after EDEN evolution in a Swimmer environment. The initialization weights follow a Gaussian distribution with a mean of zero (standard deviation 1/ $\sqrt{N}$), presenting a symmetrical bell-shaped curve. After

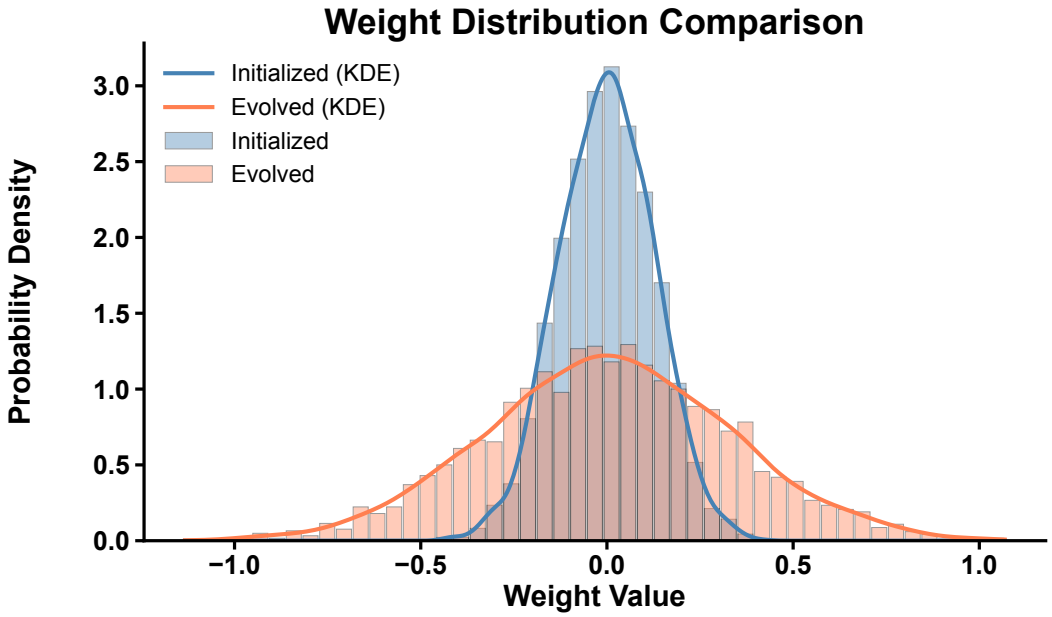

Figure 5: Comparison of neuronal connection weight distribution before and after training.

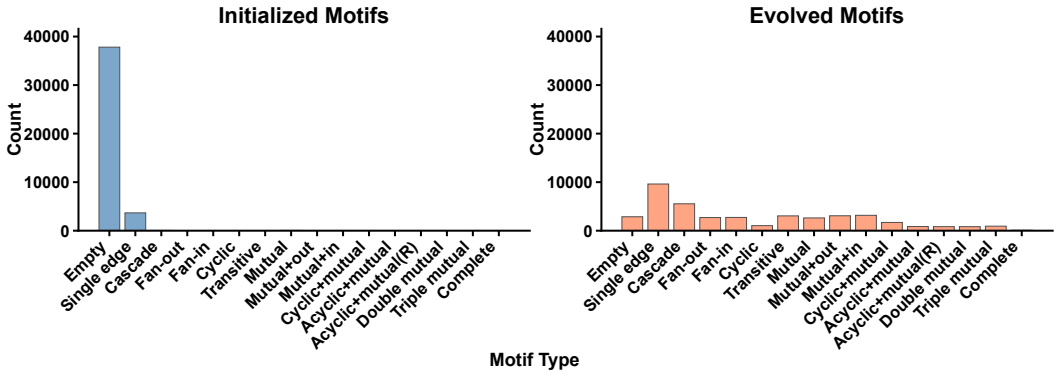

Figure 6: Statistical comparison of 16 network models before and after training.

evolutionary optimization, the weight distribution undergoes significant shift and reshaping, with the peak moving towards the negative value area and the distribution tending to flatten. This indicates that the evolutionary process, by selectively enhancing specific connections and suppressing redundant connections, enables the network to form a non-random structured weight configuration. The further smoothing of the kernel density estimation (KDE) curve reveals the systematic differences in density peak, deflection, and tail features distributed after initialization and evolution, reflecting the adaptive adjustment of connection strength during the process of network functional specialization.

Figure 6 quantified the abundance differences of 16 basic connection motifs in the EDEN network before and after evolution in a Swimmer environment through a triadic census. In the initialization network, the Empty (003) and Single edge (012) motifs are dominant, which conforms to the topological expectations of random sparse networks. In the evolved network, feedback loop modalities such as Cyclic (030C) and Cyclic+mutual (120C), as well as feedforward structure modalities such as Transitive (030T) and Cascade (021C), are significantly enriched. This modular body indicates that the evolutionary process preferentially shapes a functional topological structure that includes local cyclic loops and hierarchical feedforward pathways, providing a structural basis for information integration, temporal memory, and dynamic stability in neural computing

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: We outline the scope and contributions of our work in the last paragraph of Introduction.

2. **Limitations**

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

   Answer: [Yes]

   Justification: Limitations are discussed in the end of **Discussion** section.

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 theoretical results.

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: We provide enough details about the model and experimental setups to reproduce our results, in the main text and appendix.

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: The dataset is publicly accessible. Our code is openly available.

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: See **Method** and **Experiment** sections.

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 performances of each runs in evolutionary algorithm are presented analogously to error bars in figure 2 and **??**, and are reported in table **??**, which contains the main numerical results of the work.

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: Justification: See the **Deployment environment** in the last paragraph of Experiment section.

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: We have reviewed the NeurIPS Code of Ethics.

10. **Broader impacts**

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

    Answer: [Yes]

    Justification: The impact of the work is discussed in **Introduction** and **Discussion** sections.

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: We do not judge our model to have a high risk of misuse.

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 existing models and dataset are properly credited, and accessed with the proper license.

13. **New assets**

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

    Answer: [N/A]

    Justification: We do not introduce any new assets.

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: No human subjects were involved in this work.

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: No human subjects were involved in this work.

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: [N/A]

    Justification: Our work does not involve LLMs anywhere.