Evolving Spatially Embedded Recurrent Spiking Neural Networks for Control Tasks

Alexandru Vasilache [1,2] ✉, Jona Scholz [2], Yulia Sandamirskaya [3], and Jürgen Becker [2]

[1] FZI Research Center for Information Technology, Karlsruhe, Germany
[2] Karlsruhe Institute of Technology, Karlsruhe, Germany
[3] ZHAW Zurich University of Applied Sciences, Wädenswil, Switzerland
vasilache@fzi.de

Abstract. Spiking neural networks (SNNs), inspired by biological brains, use discrete spikes for communication, offering potential advantages in energy efficiency and temporal processing. These properties make them attractive for low-power, real-time control, but optimizing their structure and parameters is challenging. This work investigates the impact of spatial embedding on recurrent SNN performance and efficiency in continuous control tasks. We evolve SNNs with neurons positioned in a 3D Euclidean space, where connection probabilities and strengths decrease with distance. A genetic algorithm optimizes neuron parameters, connection weights, and network topology. Evaluating various spatial embeddings (none, 1D, 2D, and 3D) across multiple reinforcement learning environments, we find that spatially embedded networks outperform non-embedded counterparts within our framework. We also find that the 2D embedding generally achieves the best performance. Spatial embedding also leads to highly sparse networks, with over 95% of weights being zero. Compared to state-of-the-art deep reinforcement learning models, our evolved SNNs achieve similar performance on simpler tasks using as little as 1.46% of the non-zero weights. A performance gap on more complex tasks suggests the need for future research on larger, spatially embedded networks. These findings show the potential of spatial embedding for designing efficient SNNs for neuromorphic control.

Keywords: Spiking Neural Networks (SNNs) · Topology Optimization · Spatial Embedding · Genetic Algorithms · Control Tasks

1 Introduction

Spiking Neural Networks (SNNs) SNNs are a type of artificial neural network inspired by the biological nervous system. Unlike conventional Artificial Neuronal Networks (ANNs), which rely on continuous activation values, SNNs utilize discrete events called spikes to transmit information between neurons [13]. This event-driven nature, coupled with the statefulness and temporal dynamics of spiking neurons, enables sparse activations, making SNNs promising

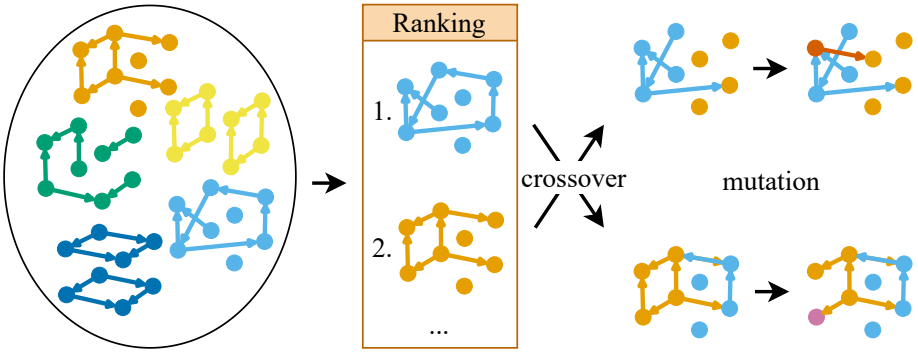

Fig. 1. Conceptual visualization of the proposed framework. Neurons are arranged in an n-dimensional grid. The method evaluates a population of networks and ranks them. High-ranking genes produce modified duplicates through crossover and mutation.

candidates for energy-efficient, low-latency computation. Recent advancements in training algorithms and neuromorphic hardware [16] further bolster the potential of SNNs for various machine learning tasks, including robot control.

SNNs for Control Tasks Research on Neural Networks (NNs) for control falls broadly into two categories: Evolutionary Algorithms (EAs) and Reinforcement Learning (RL) strategies. While both approaches show promise, SNNs present a unique challenge: effectively tuning numerous neuron-specific parameters (e.g., firing thresholds, time constants), potentially leading to smaller, more efficient networks that achieve better performance. This is where gradient-free EAs offer a distinct advantage over RL methods, which are typically limited to adjusting only synaptic weights through gradient-based methods.

EAs draw inspiration from natural selection to directly optimize all SNN parameters. For instance, Qiu et al. [22] used the NEAT algorithm [26] to evolve a recurrent spiking controller on a pole balancing task. Other works have explored EAs for optimization of various aspects of SNNs for control [23] [12] [30], highlighting their potential for such tasks.

RL methods, conversely, train SNNs by interacting with the environment and learning from rewards, inspired by natural learning. Recent advances focus on increasing training efficiency and leveraging temporal information processing capabilities. For example, Zanatta et al. [33] developed a framework for faster SNN training with Proximal Policy Optimization (PPO) [24]. Other studies have explored incorporating learning mechanisms like R-STDP [11] and temporal coding [31], and addressing challenges like domain adaptation [3].

Topology Optimization of SNNs Neural architecture search (NAS) [7], while successful in optimizing ANN architectures, is still nascent for SNNs. Existing approaches often adapt NAS techniques from ANNs, neglecting the unique tem-

poral dynamics of SNNs and primarily focusing on feedforward architectures [15] [9] [14] [20], which may limit the discovery of SNN-optimized architectures. Recurrent SNNs, commonly referred to as Liquid State Machines (LSMs) [34], offer a biologically plausible alternative to NAS for topology optimization [18] and have been proven effective for robot control tasks [17]. Beyond optimizing the overall architecture, connection optimization adjusts SNNs during training, pruning less important connections and growing new ones [21], [22], [25], [30]. Notably, Wang et al. [30] demonstrate the effectiveness of evolving connection probabilities in SNNs with sparse 1-bit connections on robotic locomotion tasks.

Spatial Embedding of Neural Networks (NNs) In a study by Deb et al. [5], weights of different network types were reshaped into 2D cortical sheets. Through the use of a topographic loss the networks learned a shape that exhibited higher weight sparsity and better parameter efficiency. Elbrecht and Schuman [6] used an evolutionary algorithm to evolve Compositional Pattern Producing Networks (CPPs) that define the connectivity, weights, delays, and neuron thresholds of SNNs whose neurons are arranged in a 2D euclidean space. Achterberg et al. [2] show that embedding Recurrent Neural Networks (RNNs) within a 3D Euclidean space and applying biophysical constraints during training leads to the emergence of brain-like characteristics, such as modularity and energy-efficient coding. Furthermore, the structure of spatially embedded networks may allow for more efficient implementations on neuromorphic chips, by aligning the physical hardware substrate more closely with the algorithm. This can be seen in work by Bogdan et al. [4], who implemented a 2D SNN on the SpiNNaker neuromorphic platform, where neurons were placed on a grid and connection rules were distance-dependent. Using Spike-Timing-Dependent Plasticity (STDP) the connection weights were evolved over time and pruned based on their magnitude. To our knowledge, our work is the first to compare the impact of different spatial embeddings on the performance SNNs.

2 Methods

For training and evaluation, we used the Gymnasium library [29], which provides standard RL environments like MuJoCo [28], ideal for testing control algorithms.

Each environment has a unique observation space, with parameters ranging from joint positions and angles to velocities, spanning a range of $(-\infty, +\infty)$, but typically within a limited range (e.g., $[0, 10]$). To provide meaningful network inputs, we determine effective ranges through multiple random runs, normalizing observations between 0 and 1. These normalized values are passed to the hidden neuron layer after being multiplied by the input connections.

The network's output is decoded by aggregating the dendritic currents (sum of synaptic inputs) of output neurons over multiple timesteps (n_{ts}). Outputs are normalized using a `sigmoid` function and scaled to fit action bounds, mapping neuronal activity to control actions.

The hidden population in the 3D SNN is implemented with a single layer of LIF neurons with sparse recurrent connections. The connectivity is based on the spatial arrangement of neurons, as detailed in Section 2.1 Network Representation. The network runs for a set number of timesteps (n_{ts}) per environment step, allowing for input integration. Neuron states reset only at the end of each environment episode, preserving information across steps.

2.1 Network Representation

Euclidean Space Encoding Neurons in the recurrent layer are assigned fixed positions in an n-dimensional grid. These positions are used in the evolution process, to determine the connection probability between two neurons in newly generated networks. It is based on the Euclidean distance d_{ij} between them:

$$P(w_{ij} \neq 0) = \begin{cases} e^{-1}, & \text{if } d_{ij} = 0 \\ e^{-d_{ij}^2}, & \text{otherwise} \end{cases} \tag{1}$$

This spatial dependence fosters locally connected networks. A distance-based scaling factor f_{ij}, ranging from 0 to 1, attenuates long-range connection strength:

$$f_{ij} = \begin{cases} 1.0, & \text{if } d_{ij} = 0 \\ (e^{-d_{ij}+1})^2, & \text{otherwise} \end{cases} \tag{2}$$

Input and output neurons are conceptually placed outside of the hidden recurrent network. The distance from input/output neurons (n_i) to any recurrent layer neuron (n_j) is artificially set to 1, ensuring a connection probability of e^{-1}, and no weight scaling.

Neurons and Synapses The basic unit in our network is the Leaky Integrate-and-Fire (LIF) neuron, implemented using the Norse framework [19]. New networks have a ratio of 80% excitatory neurons (polarity of neuron i: $p_i = 1$) and 20% inhibitory ($p_i = -1$). The membrane potential $v_i(t)$ of neuron i follows:

$$\dot{v_i}(t) = \tau_{mem_i}(v_{leak_i} - v_i(t) + I_i(t)) \tag{3}$$

where τ_{mem_i} is the membrane time constant, v_{leak_i} is the leak potential, and $I_i(t)$ is the synaptic input current. The synaptic input current is:

$$\dot{I}_i(t) = -I_i(t)\tau_{syn_i} + \sum_j z_j(t) p_j w_{ij} f_{ij} \tag{4}$$

where τ_{syn_i} is the synaptic time constant. Each connection from presynaptic neuron j to postsynaptic neuron i has a polarity p_j, strength w_{ij}, and scaling factor f_{ij}. The outgoing value of neuron j ($z_j(t)$) is either a binary spike (hidden neurons) or a numeric value (input neurons). When $v_i(t)$ reaches the threshold v_{th_i}, the neuron spikes:

$$z_i(t) = \Theta(v_i(t) - v_{th_i}) \tag{5}$$

where Θ is the Heaviside step function. After firing, $v_i(t)$ resets to v_{res_i}. In our implementation, continuous dynamics are approximated with discrete updates:

$$\begin{aligned}
i_{\text{jump}} &= I_i(t) + \sum_j z_j(t) p_j w_{ij} f_{ij} \\
dv &= \Delta t \cdot \tau_{mem_i} \left(v_{leak_i} - v_i(t) + i_{\text{jump}}\right) \\
v_{\text{decayed}} &= v_i(t) + dv \\
di &= -\Delta t \cdot \tau_{syn_i} \cdot i_{\text{jump}} \\
I_i(t+1) &= i_{\text{jump}} + di \\
z_i(t+1) &= \Theta(v_{\text{decayed}} - v_{th_i}) \\
v_i(t+1) &= (1 - z_i(t+1)) \cdot v_{\text{decayed}} + z_i(t+1) \cdot v_{res_i}
\end{aligned}$$

All the neuron and synapse-specific parameters are optimized through evolution, with their ranges and other hyperparameters summarized in Table 1.

Table 1. Parameter Values

Variable	Value Range
v_{res}	0, 0
v_{leak}	0, 127
v_{th}	1, 1000
w	0.1, 127
τ_{mem}	10, 1000
τ_{syn}	10, 1000

Mutation	Probability
add synapse	0.01
delete synapse	0.05
change weight	0.1
change polarity	0.05
change v_{th}	0.1
change v_{res}	0.1
change v_{leak}	0.1
change τ_{mem}	0.1
change τ_{syn}	0.1

Hyperparameter	Value
$\#seeds_{train}$	3
$\#seeds_{test}$	100
n_p	100
n_g	1000
n_{ts}	64
Δt	0.001
excitatory ratio	0.8

2.2 Evolutionary Algorithm

The EA iteratively refines a population of SNNs across $n_g = 1000$ generations through selection, crossover, and mutation (Fig. 1). Individuals are ranked using a weighted combination of performance and behavioral descriptors. A curriculum learning strategy accelerates evolution and improves sample efficiency. The maximum episode length starts at 10 steps, is multiplied by 1.1 after each generation, and is clipped at 1000 steps. This allows rapid evaluation and pruning of poor initial networks, while gradually increasing episode length to enable discovery of complex, long-term behaviors as the population improves.

Initialization A population of $n_p = 100$ SNNs is initialized with parameters randomly sampled within the bounds defined in Table 1.

Evaluation Each SNN is evaluated on the control task for $\#seeds_{train} = 3$ fixed environment seeds during training and for $\#seeds_{test} = 100$ for testing. The fitness is the mean reward across all training seeds. The following behavioral descriptors (scaled between [0,1]) are also measured:

- f_1: Number of active output neurons (quantifying joint usage).
- f_2: Standard deviation of active output neuron values over time (measuring joint utilization, promoting either erratic or smooth movements).
- f_3: Connection sparsity between synapses (measuring solution efficiency).

Selection and Variation A ranking identifies $m = \lfloor \sqrt{n_p} \rfloor = 10$ elite networks and is determined by a weighted sum of the fitness score and behavioral descriptors. The weights for reward, f_3, f_1, and f_2 are 100, 10, 1, and 0.1, respectively. 90% of the next generation comprises the elites (10), a mutated copy of each elite (10) and offspring (70). The 10% rest of the pool consists of (10) randomly initialized networks. Offspring are produced by applying mutation to the result of crossover between two elites. Elite pairs for crossover are selected sequentially from a list ordered by rank: [0,1], ..., [0,m], then [1,2], ..., [1,m], and so on. The variation operators are defined as follows, based of values in Table 1:
Crossover: Combines two parent networks and creates two offspring:

- Half of the neurons from each parent are randomly selected.
- Selected neurons, along with their neuron-specific parameters (including polarities), are exchanged between parents.
- Outgoing synaptic connections (including recurrent, input-to-hidden, and hidden-to-output) of the selected neurons are swapped.

Mutation: Introduces random modifications:

- Synaptic connections: Added with probability $add\ synapse = 0.01$, removed with probability $delete\ synapse = 0.05$, or have their weight modified with probability $change\ weight = 0.1$ (within $w \in (0.1, 127)$).
- Neuron properties: Polarity is altered with probability $change\ polarity = 0.05$; threshold with probability $change\ v_{th} = 0.1$; reset potential with probability $change\ v_{res} = 0.1$; leak potential with probability $change\ v_{leak} = 0.1$; membrane time constant with probability $change\ \tau_{mem} = 0.1$; and synaptic time constant with probability $change\ \tau_{syn} = 0.1$.

Network Cleanup After crossover and mutation, neurons without input connections (excluding self-connections), neurons without output connections (excluding self-connections), and isolated sub-networks (chains of neurons not connected to input or output neurons) are removed. This ensures all neurons are functionally connected.

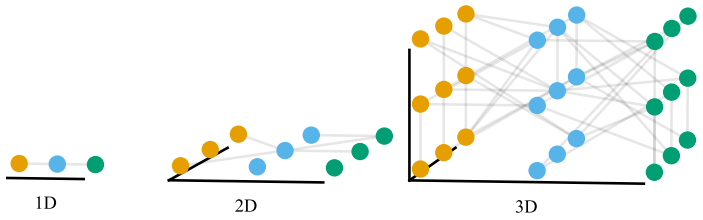

Fig. 2. Conceptual visualization of the different spatial embeddings investigated.

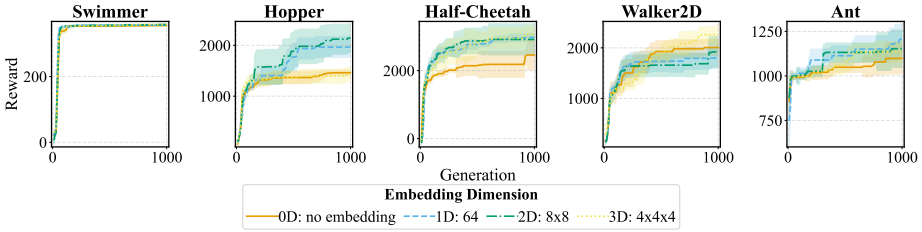

Fig. 3. Performance comparison of spatial embeddings across five environments. Plots show reward during training, evaluated on 100 unseen test seeds. Results are averaged over three independent runs; shaded regions represent standard error.

3 Experiments and Results

3.1 Spatial Embedding

To assess the impact of spatial embedding, we investigate four 64-neuron configurations, visualized in Fig. 2: 0D (no spatial embedding, connections with probability 0.1), 1D (64 neuron line), 2D (8×8 square), and 3D ($4 \times 4 \times 4$ cube). We select these embeddings as a way to investigate the effect of biological constraints on network performance and sparsity. Furthermore, higher dimensional embeddings did not present a meaningful choice, given the relatively small numbers of neurons in our network.

These embeddings are evaluated across five Gymnasium [29] environments: Swimmer-v5, Hopper-v5, Half-Cheetah-v5, Walker2D-v5, and Ant-v5. Fig. 3 shows the test fitness of the best individual of each generation for each spatial embedding and environment.

3.2 Sparsity

Spatially embedded networks show a high number of near-zero weights (Fig. 4), a consequence of the distance-based weight scaling factor f_{ij} (Equation 2).

To exploit this, we prune network weights ($w \in (0, 127)$) using absolute thresholds of 0, 0.01, 0.1, 3, 10, and 30. We then evaluate their sparsity and performance. Fig. 5 shows the results, with sparsity defined as the percentage of zero weights. The total number of weights per environment are: Swimmer: 4736, Hopper: 4992, Half-Cheetah: 5568, Walker2D: 5568, and Ant: 11328.

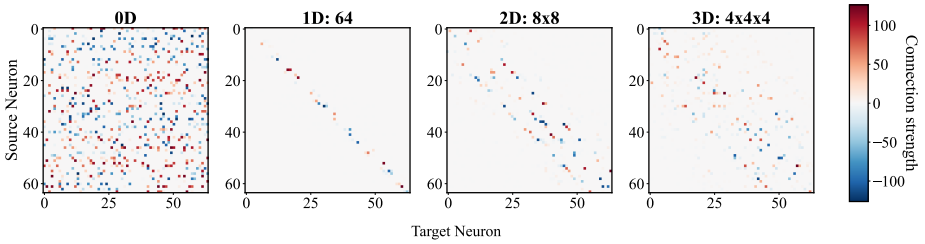

Fig. 4. Connectivity strength between hidden neurons for different dimensional embeddings. In 0D, connection probability is uniform. In nD, distance determines connection likelihood and strength. Higher dimensions allow more connections per neuron.

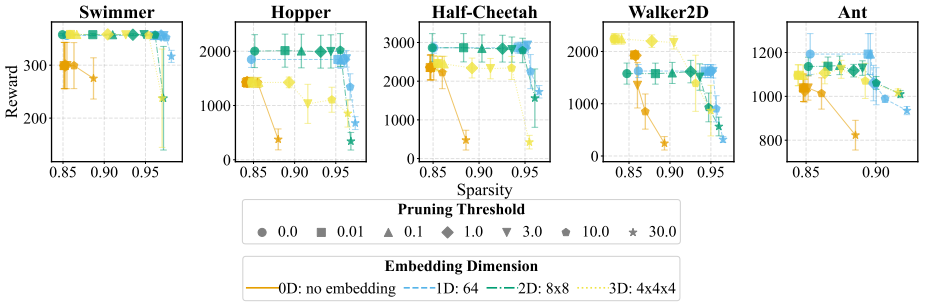

Fig. 5. Performance-sparsity plots of pruned SNNs across five environments. Markers represent different weight pruning thresholds. Sparsity is the percentage of zero weights. Error bars represent standard error over three training runs. Total weight counts are: Swimmer: 4736, Hopper: 4992, Half-Cheetah: 5568, Walker2D: 5568, Ant: 11328.

3.3 Comparison with State-of-the-Art Baselines

Table 2 compares our spatially embedded models to the best performing models ("expert") from the Farama-Minari repository [32], trained with PPO [24], SAC [8], and TQC [10], by analyzing their architectures [1] and weight counts.

Baseline models have between 9,408 and 475,392 weights, while ours have significantly fewer: 137 (Swimmer), 222 (Hopper), 703 (HalfCheetah), 694 (Walker2d), and 1208 (Ant). This corresponds to parameter reductions of 98.54% to 99.94%. Relative performance is 99.48% (Swimmer), 66.03% (Hopper), 20.81% (HalfCheetah), 35.82% (Walker2d), and 23.04% (Ant).

4 Discussion

Comparing the performance of spatial embeddings, we observe that they consistently outperformed non-embedded networks within our framework (Fig. 3). Furthermore, the 2D spatial embedding generally achieved the best performance especially in less complex environments, which may represent evidence for the similar 2D sheet-like structures observed in the brain, as cortical sheets [27].

Table 2. Comparison of Spatially Embedded SNNs vs. SOTA Baselines with standard deviation shown. Our models are embedded in 2D (Swimmer, Hopper, HalfCheetah), 3D (Walker2d), and 1D (Ant). Pruning thresholds are: 30.0 (Swimmer), 10.0 (Hopper), 0.01 (HalfCheetah), 1.0 (Walker2d), and 0.1 (Ant). "NZW" is the number of non-zero weights. "RWC" (relative weight count) and "RP" (relative performance) are expressed as a percentage of the baseline values.

Environment	**SOTA Baseline**		**Spatially Embedded**			
	NZW	**Reward**	**NZW**	**RWC**	**Reward**	**RP**
Swimmer	9,408	363.70 ± 0.06	137	1.46%	361.81 ± 0.17	99.48%
Hopper	347,392	4098.17 ± 7.83	222	0.06%	2706.10 ± 7.37	66.03%
HalfCheetah	384,256	17641.80 ± 1.96	703	0.18%	3670.43 ± 119.83	20.81%
Walker2d	359,680	6956.61 ± 0.50	694	0.19%	2491.96 ± 24.77	35.82%
Ant	475,392	5846.28 ± 43.80	1,208	0.25%	1346.85 ± 3.51	23.04%

The 1D and 3D embeddings proved especially efficient for specific environments. These preliminary findings should be confirmed by future studies through evaluation on an increased number of control environments, as the current number may be insufficient to draw definitive conclusions and to fully capture the diversity of continuous control.

Furthermore, spatially embedded networks achieved high sparsity levels (over 95%) with minimal performance impact (Fig. 5). The 1D embedding was particularly robust, allowing for pruning of weights below 3 without any performance degradation in most environments.

Table 2 reveals a substantial non-zero weight (NZW) reduction (98.54%-99.94%) compared to state-of-the-art models, highlighting the efficiency achievable through spatial embedding and pruning. However, a performance trade-off exists. While our best performing model achieved near-identical performance (99.48%) to the baseline on the simpler Swimmer environment using only 1.46% of the NZWs, more complex environments exhibited a larger performance gap (23.04% for Ant, 20.81% for HalfCheetah), which generally increased with task complexity. This suggests that while the current network size of 64 neurons is sufficient for simple tasks, larger networks are likely necessary for more complex problems. Despite this, our models still achieved meaningful performance on these complex tasks (outperforming the "simlpe" models in the Farama-Minari repository), while using less than 0.3% of the NZWs of the baselines.

A key limitation of the current study is the computational cost of the framework. Simulating larger networks, which preliminary experiments suggest would improve performance, is currently infeasible due to the implementation's inefficiency. Evaluating a single environment for five embeddings across 1000 generations and three training seeds required approximately 150 hours on a server equipped with 2x AMD EPYC 7713 64-Core Processors, 3x NVIDIA RTX A6000 48GB, 4x NVIDIA RTX 6000 48GB, and 512 GB DDR4-3200 RAM.

5 Conclusion

We introduced a novel evolutionary framework for optimizing recurrent SNNs. Instead of treating SNNs as abstract networks, we positioned neurons within a 3D Euclidean space, mirroring the physical organization of biological brains. This spatial embedding, coupled with distance-dependent connections, naturally fostered the evolution of locally connected, sparse networks. A genetic algorithm then refined not just the network's connections, but also the parameters of each neuron (such as firing thresholds and time constants), a capability often lacking in gradient-based reinforcement learning approaches.

Spatially embedded SNNs consistently surpassed the performance of non-embedded networks within our framework. Moreover, the evolved networks were remarkably sparse, with over 95% of all possible connections eliminated while maintaining the same performance. Compared to state-of-the-art deep reinforcement learning models, our SNNs achieved similar performance on simpler tasks while using as little as 1.46% of the non-zero weights (NZWs). Despite the extreme NZW reduction (some models using less than 0.1% of the baseline NZWs), with increasing task complexity, a larger performance gap emerged. This is due to the limited size of our networks, using only 64 neurons, as this work is intended as a proof-of-concept regarding the potential of evolving spatially embedded SNNs.

These findings point to spatial embedding as a candidate for the practical application of SNNs in low-power control. Future work will focus on scaling this approach to larger networks and improving the computational efficiency of the evolutionary optimization, to tackle more complex control tasks. The ultimate goal is to develop SNN-based controllers that are not only powerful but also highly efficient, ideal for deployment in real-world, resource-limited scenarios.

Disclosure of Interests The authors have no competing interests to declare that are relevant to the content of this article.

Code Availability The source code for our experiments is available on GitHub: `https://github.com/Alex-Vasilache/Spatial-SNNs`

References

1. farama-minari (Minari) (Feb 2025), `https://huggingface.co/farama-minari`, accessed on 2025-06-05
2. Achterberg, J., Akarca, D., Strouse, D.J., Duncan, J., Astle, D.E.: Spatially embedded recurrent neural networks reveal widespread links between structural and functional neuroscience findings. Nature Machine Intelligence **5**(12), 1369–1381 (Nov 2023). `https://doi.org/10.1038/s42256-023-00748-9`
3. Akl, M., Sandamirskaya, Y., Ergene, D., Walter, F., Knoll, A.: Fine-tuning Deep Reinforcement Learning Policies with r-STDP for Domain Adaptation. In: Proceedings of the International Conference on Neuromorphic Systems 2022. pp. 1–8. ACM, Knoxville TN USA (Jul 2022). `https://doi.org/10.1145/3546790.3546804`

4. Bogdan, P.A., Rowley, A.G., Rhodes, O., Furber, S.B.: Structural plasticity on the spinnaker many-core neuromorphic system. Frontiers in Neuroscience **12**, 434 (2018)
5. Deb, M., Deb, M., Murty, N.: Toponets: High performing vision and language models with brain-like topography. arXiv preprint arXiv:2501.16396 (2025)
6. Elbrecht, D., Schuman, C.: Neuroevolution of Spiking Neural Networks Using Compositional Pattern Producing Networks. In: International Conference on Neuromorphic Systems 2020. pp. 1–5. ACM, Oak Ridge TN USA (Jul 2020). https://doi.org/10.1145/3407197.3407198
7. Elsken, T., Metzen, J.H., Hutter, F.: Neural architecture search: A survey (2019)
8. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor (2018)
9. Kim, Y., Li, Y., Park, H., Venkatesha, Y., Panda, P.: Neural Architecture Search for Spiking Neural Networks (Jul 2022), arXiv:2201.10355 [cs, eess]
10. Kuznetsov, A., Shvechikov, P., Grishin, A., Vetrov, D.: Controlling overestimation bias with truncated mixture of continuous distributional quantile critics (2020)
11. Liu, Y., Pan, W.: Spiking Neural-Networks-Based Data-Driven Control. Electronics **12**(2), 310 (Jan 2023). https://doi.org/10.3390/electronics12020310
12. Lu, J., Hagenaars, J.J., de Croon, G.C.H.E.: Evolving-to-Learn Reinforcement Learning Tasks with Spiking Neural Networks (Feb 2022), arXiv:2202.12322 [cs]
13. Maass, W.: Networks of spiking neurons: the third generation of neural network models. Neural networks **10**(9), 1659–1671 (1997)
14. Man, Y., Xie, L., Qiao, S., Zhou, Y., Shang, D.: Differentiable architecture search with multi-dimensional attention for spiking neural networks. Neurocomputing **601**, 128181 (Oct 2024). https://doi.org/10.1016/j.neucom.2024.128181
15. Na, B., Mok, J., Park, S., Lee, D., Choe, H., Yoon, S.: AutoSNN: Towards Energy-Efficient Spiking Neural Networks. In: Proceedings of the 39th International Conference on Machine Learning. pp. 16253–16269. PMLR (Jun 2022), iSSN: 2640-3498
16. Neuromorphic, O.: Neuromorphic Hardware Guide, accessed on 2025-06-05
17. Pan, W., Zhao, F., Shen, G., Zeng, Y.: Multi-scale Evolutionary Neural Architecture Search for Deep Spiking Neural Networks (Nov 2023), arXiv:2304.10749
18. Pan, W., Zhao, F., Zhao, Z., Zeng, Y.: Brain-inspired Evolutionary Architectures for Spiking Neural Networks (Sep 2023), arXiv:2309.05263 [cs]
19. Pehle, C.G., Egholm Pedersen, J.: Norse-a deep learning library for spiking neural networks. Zenodo (2021)
20. Putra, R.V.W., Shafique, M.: SpikeNAS: A Fast Memory-Aware Neural Architecture Search Framework for Spiking Neural Network-based Autonomous Agents (Apr 2024), arXiv:2402.11322 [cs]
21. Qi, Y., Shen, J., Wang, Y., Tang, H., Yu, H., Wu, Z., Pan, G.: Jointly Learning Network Connections and Link Weights in Spiking Neural Networks (Jul 2018). https://doi.org/10.24963/ijcai.2018/221, pages: 1603
22. Qiu, H., Garratt, M., Howard, D., Anavatti, S.: Evolving Spiking Neural Networks for Nonlinear Control Problems. In: 2018 IEEE Symposium Series on Computational Intelligence (SSCI). pp. 1367–1373 (Nov 2018). https://doi.org/10.1109/SSCI.2018.8628848, arXiv:1903.01180 [cs]
23. Rafe, A.W., Garcia, J.A., Raffe, W.L.: Exploration Of Encoding And Decoding Methods For Spiking Neural Networks On The Cart Pole And Lunar Lander Problems Using Evolutionary Training. In: 2021 IEEE Congress on Evolutionary Computation (CEC). pp. 498–505. IEEE, Kraków, Poland (Jun 2021). https://doi.org/10.1109/CEC45853.2021.9504921

24. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. CoRR **abs/1707.06347** (2017)
25. Shen, J., Xu, Q., Liu, J.K., Wang, Y., Pan, G., Tang, H.: ESL-SNNs: An Evolutionary Structure Learning Strategy for Spiking Neural Networks. Proceedings of the AAAI Conference on Artificial Intelligence **37**(1), 86–93 (Jun 2023). https://doi.org/10.1609/aaai.v37i1.25079
26. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary Computation **10**(2), 99–127 (2002). https://doi.org/10.1162/106365602320169811
27. Tax, C.M., Westin, C.F., Haije, T.D., Fuster, A., Viergever, M.A., Calabrese, E., Florack, L., Leemans, A.: Quantifying the brain's sheet structure with normalized convolution. Medical image analysis **39**, 162–177 (2017)
28. Todorov, E., Erez, T., Tassa, Y.: Mujoco: A physics engine for model-based control. In: 2012 IEEE/RSJ international conference on intelligent robots and systems. pp. 5026–5033. IEEE (2012)
29. Towers, M., Terry, J.K., Kwiatkowski, A., Balis, J.U., Cola, G.d., Deleu, T., Goulão, M., Kallinteris, A., KG, A., Krimmel, M., Perez-Vicente, R., Pierré, A., Schulhoff, S., Tai, J.J., Shen, A.T.J., Younis, O.G.: Gymnasium (Mar 2023). https://doi.org/10.5281/zenodo.8127026
30. Wang, G., Sun, Y., Cheng, S., Song, S.: Evolving Connectivity for Recurrent Spiking Neural Networks. In: Advances in Neural Information Processing Systems. vol. 36, pp. 2991–3007. Curran Associates, Inc. (2023)
31. Wu, G., Liang, D., Luan, S., Wang, J.: Training Spiking Neural Networks for Reinforcement Learning Tasks With Temporal Coding Method. Frontiers in Neuroscience **16**, 877701 (Aug 2022). https://doi.org/10.3389/fnins.2022.877701
32. Younis, O.G., Perez-Vicente, R., Balis, J.U., Dudley, W., Davey, A., Terry, J.K.: Minari (Sep 2024). https://doi.org/10.5281/zenodo.13767625
33. Zanatta, L., Barchi, F., Manoni, S., Tolu, S., Bartolini, A., Acquaviva, A.: Exploring Spiking Neural Networks for Deep Reinforcement Learning in Robotic Tasks: A Comparative Study (Jun 2024). https://doi.org/10.21203/rs.3.rs-4470688/v1
34. Zhou, Y., Jin, Y., Ding, J.: Surrogate-Assisted Evolutionary Search of Spiking Neural Architectures in Liquid State Machines. Neurocomputing **406**, 12–23 (Sep 2020). https://doi.org/10.1016/j.neucom.2020.04.079