Neuromorphic Evolution Spiking Networks in the Snake Environment

Lucca Forrest[1, 2, *]

[1] *Department of Physics, University of North Carolina at Chapel Hill*
[2] *Department of Physics, University of Oxford*

(Dated: November 26, 2025)

We present a neural architecture derived from first principles of population dynamics rather than gradient-based optimization. The system is initialized with minimal structure and evolves through genetic encoding, reproduction, and mutation. Neurons are binary integrators with leakage and recurrence, producing time-dependent activity that enables internal memory and sequential decision-making. Learning emerges from evolutionary selection on accuracy and reaction time, not from forward- or backpropagation. We demonstrate this framework on the classic Snake game, where networks must rapidly acquire effective strategies by evolving decision pathways rather than memorizing patterns. Unlike conventional deep learning, the approach prioritizes adaptability and speed: output neurons compete under threshold-based firing rules, and indecision can resolve through randomized or default choice, maintaining evolutionary pressure toward decisive behavior. The resulting networks scale in complexity as tasks demand, combining structural growth with computational efficiency through vectorized operations over populations. This work suggests that evolutionary neural systems can achieve rapid mastery of sequential decision-making tasks, while raising new questions about interpretability, safety, and the boundaries of unsupervised network growth. While our initial experiments use Snake as a proof of concept, extending this approach to complex strategic domains like chess is discussed as a future direction.

I. ARTIFICIAL AND BIOLOGICAL NEURONS

A. Perceptron: neural networks at the neuron level

At the most fundamental level, an artificial neuron is a mathematical function that transforms a set of inputs into a single output. Each input x_i is multiplied by an associated weight w_i, summed together with a bias term b, and passed through a nonlinear activation function σ:

$$z = \sum_{i=1}^{n} w_i x_i + b, \qquad y = \sigma(z), \tag{1}$$

where z represents the pre-activation value, and y is the neuron's output. The activation function introduces nonlinearity, enabling networks to approximate complex mappings rather than just linear transformations.

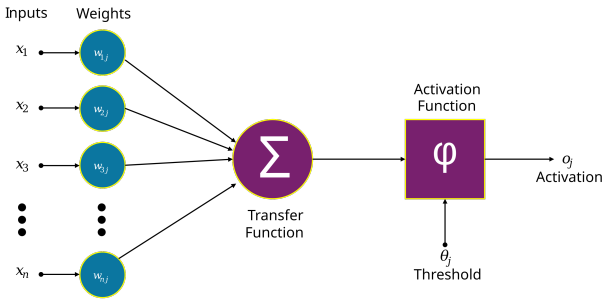

FIG. 1. Schematic of a single artificial neuron. Inputs x_i are weighted, summed, and transformed by an activation function to produce an output y.

The output of a neuron is then passed forward as input to other neurons. When many neurons are organized into layers, the outputs of one layer become the inputs of the next. In this way, neurons combine to form hierarchical feature representations.

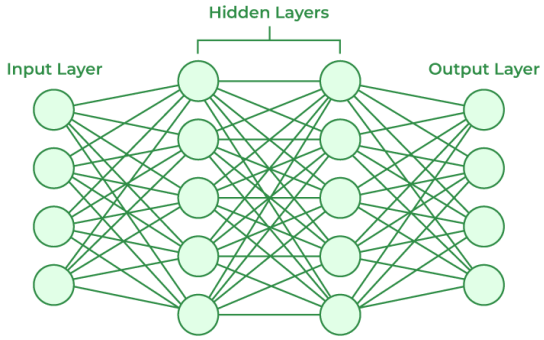

FIG. 2. Artificial neurons combined into layers and stacked into a multi-layer network. The outputs of one layer serve as inputs to the next.

This weighted-sum neuron with nonlinear activation is the foundation of most modern architectures, from convolutional networks to transformers. In the context of any complex decision task, it provides the basic computational unit for agents to map states to evaluations or actions.

B. Biological neurons

Biological neurons are structurally far more complex than the artificial perceptrons used in modern neural net-

* lucca˙forrest@unc.edu

works. A single neuron consists of dendrites, a soma (cell body), an axon, and synapses that transmit chemical signals via neurotransmitters. The dendrites receive thousands of inputs of different types, which are integrated in the soma and may trigger an electrical impulse along the axon. This output spike then influences downstream neurons through synaptic connections.

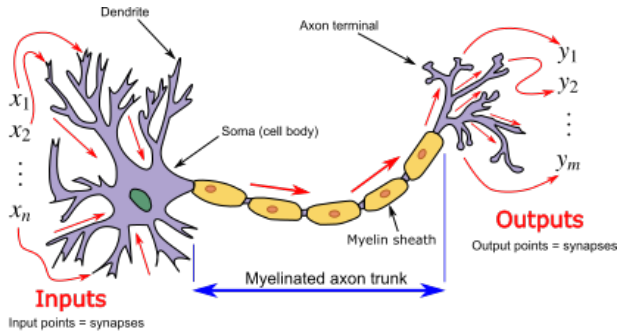

FIG. 3. Schematic of a biological neuron, showing dendrites (inputs), soma (integration), axon (signal propagation), and synapses (outputs). Excitatory, inhibitory, and modulatory neurotransmitters shape the signal before an action potential is fired.

C. Neural input and output

For comparison with artificial neurons, it is useful to oversimplify the complex physiology of biological neurons into three dimensions of input: strength, timing, and type. Strength corresponds to the amount of neurotransmitter released at the synapse; timing reflects when signals arrive relative to one another; and type refers to whether a signal is excitatory, inhibitory, or modulatory. Artificial neurons, in contrast, generally operate along a single dimension of input: strength, expressed as a weighted sum.

The outputs also differ substantially. Biological neurons do not produce continuous values; rather, they fire discrete spikes once the membrane potential crosses a threshold. These spikes are all-or-none events, akin to a binary switch. Signal strength is not encoded in the amplitude of a spike (which is fixed), but in the frequency of spikes (rate coding) and in the precise timing relative to other neurons (temporal coding). In contrast, artificial neurons typically output continuous values that are passed to the next layer without explicit temporal structure.

D. Alignment of biological and artificial neurons

The history of activation functions in machine learning highlights how artificial neurons evolved under mathematical constraints rather than biological realism. Early models (e.g. McCulloch–Pitts neurons) employed step functions, binary thresholds where a unit either fired (1) or did not (0). While this resembled the all-or-none character of biological spikes, step functions are not differentiable, which prevented effective training using gradient descent. This limitation meant only very shallow networks could be trained.

The next era introduced smooth, differentiable functions such as the sigmoid and hyperbolic tangent (tanh). These allowed backpropagation and gradient-based learning, enabling deeper architectures. However, sigmoid and tanh functions suffered from the vanishing gradient problem: as depth increased, gradients often shrank toward zero, hindering effective weight updates in very deep networks.

Modern deep learning largely relies on the rectified linear unit (ReLU), defined as $f(x) = \max(0, x)$, and its variants (Leaky ReLU, GELU, Swish). ReLU reintroduced a form of sparse, switch-like activation while remaining piecewise linear and differentiable almost everywhere. This yields both computational efficiency and an implicit sparsity (many neurons inactive for a given input), somewhat echoing the brain's efficiency while still being amenable to gradient-based optimization.

E. Computation and scale

Despite their capabilities, modern artificial neural networks require substantial computational resources. The human brain operates at roughly 20 W, yet contemporary AI models demand many orders of magnitude more power. Large-scale models are typically trained on clusters of GPUs or TPUs drawing tens or hundreds of kilowatts or more. These considerations highlight the potential benefits of biologically inspired architectures, which might offer improved energy efficiency and scalability by leveraging principles of sparse, event-driven computation like the brain.

F. Speed and precision

While artificial neural networks incur high energy costs, they offer substantial advantages in speed, precision, and scalability. Unlike biological systems, which rely on relatively slow electrochemical signaling (on the order of milliseconds), artificial systems operate at electronic speeds (nanoseconds). Large-scale models can perform parallel computations across thousands of processors, enabling them to process vast datasets and perform inference at speeds far beyond human capability. Additionally, artificial systems exhibit reproducibility and numerical precision that biological neurons lack, allowing for controlled experimentation, exact replication, and deployment at industrial scales.

G. Backpropagation vs. biological learning

Modern artificial neural networks rely on backpropagation, a supervised learning algorithm in which an explicit error signal is propagated backward through the network to adjust weights via gradient descent. This process requires vast quantities of labeled data and iterative passes, and the network's architecture (graph of connections) remains fixed during training.

In contrast, human learning is more flexible, efficient, and context-aware. Humans can generalize from few examples, learn continuously from unstructured experiences, and adapt both cognitive strategies and neural pathways over time. Learning in the brain is often reinforced through feedback, memory, attention, and reward, rather than through explicit gradient-based updates on every connection. Furthermore, the human brain exhibits structural plasticity: not only do synaptic strengths change, but new synapses and neurons can form while others wither. This means the very architecture of biological neural networks can evolve with experience—a feature largely absent in current deep learning systems, which typically have a fixed topology defined ahead of time.

H. Work on biologically inspired models

Biologically inspired models seek to move beyond the abstractions of conventional artificial neurons by incorporating mechanisms that more closely resemble the structure and dynamics of real neural tissue. Notable among these are spiking neural networks (SNNs), which encode information using discrete spikes over time, mimicking the sparse, event-driven signaling of biological neurons. Unlike standard artificial networks that propagate continuous activations each timestep or layer, SNNs use membrane potentials and threshold-based firing, offering energy-efficient and temporally sensitive computation.

Additional efforts model more intricate biological features such as dendritic computation, local learning rules (e.g. Hebbian plasticity), cortical microcircuits, and neuromodulation. Some research explores neurons that output a vector of values, where each component might correspond to a different "neurotransmitter" type, with downstream neurons responding selectively to different components. Neuromodulated neural networks explicitly incorporate special units that alter the dynamics or plasticity of other neurons, inspired by how neuromodulators in the brain influence learning and attention.

Vector Symbolic Architectures (VSA) and related approaches use high-dimensional vectors to represent symbolic structures and operations, echoing hypotheses about distributed representations in cortex. These biologically inspired ideas are increasingly supported by specialized neuromorphic hardware platforms such as Intel's Loihi and IBM's TrueNorth, which use asynchronous circuits and event-driven design to approximate brain-like efficiency.

Our approach takes inspiration from this line of work, combining spiking neuron dynamics with an evolutionary, structurally plastic architecture that can grow in complexity over generations.

II. NETWORK ARCHITECTURE FIRST PRINCIPLES

We now formalize the design principles underlying our architecture. These principles span neuron-level dynamics, population-level learning, and meta-level considerations.

A. Neuron-level dynamics

Input neurons. Input neurons pass continuous-valued sensory signals into the network at time $t = 0$ for each decision episode. They do not spike; instead, their activations are clamped to the input feature values and remain fixed throughout the episode. The number of input neurons matches the dimensionality of the input, $n_{\text{in}} = d_{\text{in}}$.

Hidden and output neurons (spiking). All internal neurons are binary spiking units. They integrate incoming activity over time with leak (decay) and fire a discrete spike when their membrane potential exceeds a threshold. Upon firing, a neuron's potential resets to a baseline value. This on/off spiking behavior introduces nonlinearity and enables temporal dynamics and internal memory. The number of output neurons equals the number of possible decisions or classes, $n_{\text{out}} = d_{\text{out}}$.

Activation accumulation. Neurons accumulate weighted input from connected units at each time step. A neuron's membrane potential increases with incoming spikes and decays by a factor $0 < \gamma < 1$ each step.

Thresholds. Each spiking neuron has a firing threshold θ. When its membrane potential crosses this threshold, the neuron emits a spike and its potential is reset.

Leakage. Between inputs, a neuron's membrane potential is multiplied by a decay factor at each time step, representing passive loss of activation over time. This enforces temporal dependence and urgency: if a neuron does not reach threshold quickly, its accumulated potential leaks away, requiring sustained input to eventually fire.

Recurrence. Hidden neurons are richly interconnected. The network can contain cycles among hidden units, allowing feedback loops. This recurrence means a neuron's spike can influence others (or itself via indirect paths) at later time steps, enabling oscillatory or persistent activity patterns that serve as internal memory.

Discrete time. Computation unfolds over a series of discrete steps. At each step, all neurons update their potentials based on inputs received and decay, possibly spike if threshold is exceeded, and then proceed to the

next step. The network's output is thus a process in time rather than a single instant.

Decision rule. The network's decision for a given input is determined by the first output neuron to fire. Output neurons compete to reach threshold; the one that spikes first "wins" and its associated action or class is taken as the network's output. If none fire within a maximum number of steps, the network may output a default action (e.g. the maximal output at the final step) or be treated as undecided.

B. Population-level learning

Learning without backpropagation. There is no forward/backward gradient-based learning. Instead, learning occurs at the population level: a population of networks is maintained and improved via evolutionary algorithms rather than adjusting synapses via error gradients in a single network.

Genetic encoding. Each network's parameters (connection weights, bias terms, and connectivity structure) are encoded in a "genome" comprising weight matrices, bias vectors, and a binary connectivity mask.

Selection. The population is evaluated on the task (e.g. playing games). Each network's fitness is determined by its performance (score, survival, decision speed). The highest-performing networks are considered parents for the next generation.

Reproduction and mutation. New networks are generated from elites by copying and then applying random mutations to their genomes: perturbing weights and biases and occasionally adding new connections where allowed by the mask.

Initialization and structural plasticity. The initial population starts with minimal structure: small networks with mostly random connectivity and weights. Over generations, mutation can introduce new hidden–hidden and hidden–output connections. Thus, networks can grow in effective complexity if the task rewards it.

Evolution vs. optimization. Adaptation occurs through natural selection on a population, rather than through gradient descent on a single network. There is no explicit gradient or loss function at the synapse level—only the fitness of whole networks, which is a weaker but more global signal.

C. Meta considerations

Computation. Evolving a population with time-step simulations for each network can be expensive. We therefore use vectorized operations to update many networks in parallel and stop simulating each network as soon as it has produced a decision.

Black-box concerns. Because learning occurs via random variation and selection, and networks may grow in idiosyncratic ways, the resulting solutions can be opaque. Decision pathways are not easily interpretable; dynamics are entangled in time and recurrence.

Safety. In this study, our evolved networks are confined to toy domains (Snake). Applying open-ended evolutionary strategies to real-world tasks raises safety concerns: evolution can exploit loopholes in reward functions and produce unexpected behaviors. Understanding and constraining such systems is an important area for future work.

III. NETWORK METHODS

A. Neuronal types

The network employs two distinct classes of neurons with different roles and behaviors:

Input neurons (continuous-valued). Input neurons directly encode external stimulus values into the network. They do not use a spiking threshold; each input neuron's activation is set to a normalized value corresponding to some aspect of the sensory input and remains clamped throughout the decision cycle. The number of input neurons equals the dimensionality of the input data, $n_{\text{in}} = d_{\text{in}}$.

Hidden and output neurons (spiking units). All internal processing neurons (hidden and output) are spiking neurons governed by an accumulate-and-fire rule. Each such neuron integrates input from connected neurons into a membrane potential that decays over time and fires a spike when a threshold is exceeded. Hidden neurons provide internal computation and can be recurrently connected, while output neurons correspond to discrete actions or classes. The number of output neurons equals the number of possible choices, $n_{\text{out}} = d_{\text{out}}$. In Snake we use three outputs (straight, left, right).

This separation yields a hybrid design: inputs are analog, but internal processing and outputs are digital spikes.

B. Divergence from a standard XOR network

It is useful to illustrate how the architecture differs from a conventional network via the classic XOR problem. XOR (exclusive-OR) outputs true if and only if its two binary inputs differ (Table I).

Input A	Input B	Output $A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

TABLE I. Truth table for the XOR operation.

A single perceptron cannot represent XOR, since it is

not linearly separable. The standard solution is a 2–2–1 network: two inputs, two hidden neurons, and one output neuron producing a scalar interpreted as $\mathbb{P}(\textit{output} = 1)$.

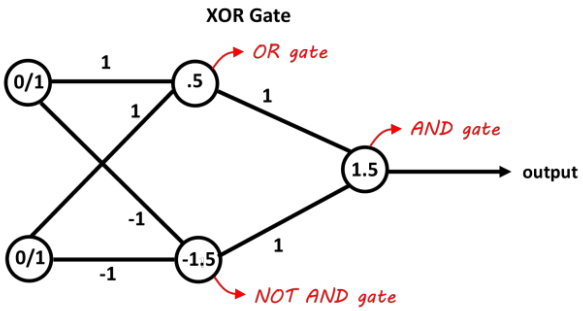

FIG. 4. Illustrative XOR networks. Circles indicate continuous-valued neurons, diamonds spiking neurons. A standard 2–2–1 network uses a single continuous output; this is replaced with two competing spiking outputs whose race to threshold determines the class.

We modify this design in two key ways (schematically represented in Fig. 4):

(a) **Spiking neurons.** Hidden and output neurons are spiking, whereas only the input layer is continuous. This mirrors cortical neurons communicating via discrete action potentials.

(b) **Competing outputs.** Instead of a single probabilistic output, our network uses two spiking output neurons V_1 and V_2, each corresponding to one of the two classes. These neurons compete to fire; whichever spikes first determines the XOR output.

Through this XOR lens notice that this network introduces time and competition even into simple computations. A traditional network instantly computes a probability; our network can wait a few steps until one output spikes, with ambiguous inputs yielding delayed or absent spikes.

C. Spiking dynamics and leakage

A typical leaky integrate-and-fire (LIF) model tracks a membrane potential $V(t)$ obeying

$$\tau \frac{dV}{dt} = -\left(V(t) - V_{\text{rest}}\right) + I(t), \tag{2}$$

where τ is a membrane time constant, V_{rest} a resting potential and $I(t)$ an input current. When $V(t)$ exceeds threshold V_{th}, the neuron emits a spike and V is reset.

In our discrete-time implementation the activations of all neurons are collected in a vector $\mathbf{a}^{(t)} \in \mathbb{R}^N$, where $N = n_{\text{in}} + n_{\text{hid}} + n_{\text{out}}$. Given weight matrix W and bias vector $\mathbf{B}$, the update at each recurrent step is

$$\Delta^{(t)} = \mathbf{a}^{(t)} W + \mathbf{B}, \tag{3}$$

$$\mathbf{a}^{(t+1)} = \gamma \mathbf{a}^{(t)} + \Delta^{(t)}, \tag{4}$$

with decay factor $0 < \gamma < 1$. Input neurons are clamped back to the external input after each update, hidden activations are passed through tanh, and outputs remain linear but are monitored for threshold crossings.

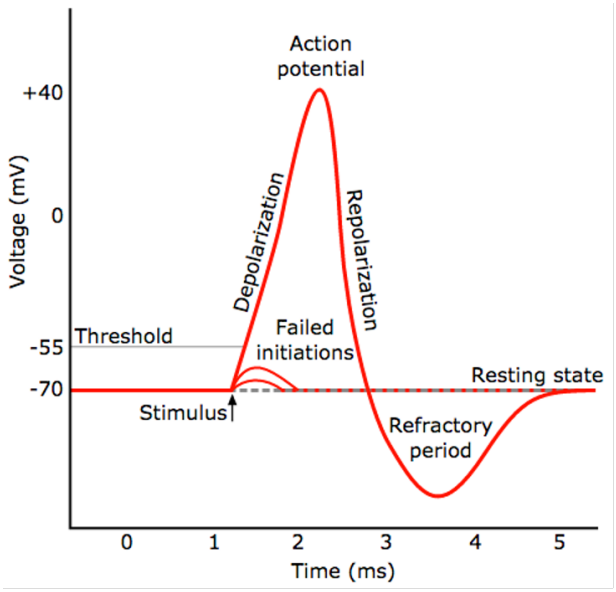

FIG. 5. Example membrane potential trace showing discrete spikes when the threshold is crossed. Between spikes the potential leaks back toward rest.

D. Rat left–right analogy

To build intuition for the decision mechanism, consider a rat in a T-maze deciding to turn left or right. As it approaches the fork, it accumulates evidence (smells, memories of rewards). Neurons in its brain may show increasing activity favoring "left" versus "right" as it deliberates, until one pool of neurons crosses a threshold and triggers the behavioral response. This is an "accumulator" model of decision-making.

In this network, output neurons play the role of these accumulators. Each output neuron integrates evidence from hidden neurons. Over recurrent time steps, their potentials rise; whichever output reaches threshold first spikes and determines the decision. Decisions thus have variable timing: clear evidence yields fast spikes, while ambiguous inputs yield slower or absent decisions.

E. Indecision handling

What if neither decision neuron reaches threshold? In biological terms, the rat may freeze or hesitate. In our network, we explicitly allow for an "undecided" outcome if no output fires in the allotted time. Two scenarios arise:

1. Hidden activity ceases before any output spikes; the network effectively gives up.
2. A maximum number of steps $T_{\max}$ is reached without threshold crossing.

In general we can treat indecision as a partial failure (for example, assigning intermediate fitness compared to wrong vs. correct decisions), encouraging evolution to develop decisive circuits. In the specific Snake implementation below, indecision is extremely rare for trained networks, and we resolve it by defaulting to the maximal output at the final step (effectively a soft argmax decision).

F. Connectivity rules

Source ↓ Target →	Input	Hidden	Output
Input	0	1	0
Hidden	0	1	1
Output	0	0	0

TABLE II. Connectivity constraints mask. A value of 1 denotes an allowed projection (source → target). Self-loops are prohibited, but recurrent connections between distinct hidden neurons remain permitted.

Most rules are straightforward and consistent with both biological plausibility and computational design. Two cases merit deeper justification: the disallowance of Input→Output and the allowance of dense Hidden→Hidden recurrence.

1. Rationale for disallowing Input→Output

In biological brains, sensory input usually passes through multiple processing stages before influencing motor output. There is no direct wire from retina to hand muscles; intermediate areas integrate and transform the signal. Inspired by this, we prevent direct Input→Output connections. If they were allowed, evolution might discover simple hard-coded stimulus–response reflexes, short-circuiting the development of internal representations and memory in the hidden layer. Forcing all decisions to be mediated by hidden neurons encourages richer internal computation.

2. Rationale for allowing dense Hidden→Hidden recurrence

We allow fully recurrent connectivity among hidden neurons (excluding self-loops). This maximizes the space of possible dynamics: the hidden layer can form oscillators, memory loops, multi-stable toggles, and other motifs.

Alternatives were considered and rejected:

- Disallowing all Hidden→Hidden connections would reduce the model to a feedforward network unrolled in time, undermining our first principles of recurrence and temporal computation.
- Restricting recurrence to an acyclic pattern would be biologically unrealistic and cumbersome to enforce under structural mutation.

Dense recurrence does introduce possibilities for unstable feedback. However, leak, thresholds, and inhibitory weights provide damping, and networks with pathological dynamics tend to perform poorly and are eliminated by selection.

IV. POPULATION METHODS

A. Vectorized simulation of populations

Evolving a population of networks with time-step simulations for each decision is computationally intensive. A naive approach would simulate each network independently until it decides or hits the step limit. Instead, we use vectorized simulation: many networks are updated in parallel using batched matrix operations.

Conceptually, this is like running many rats through many mazes simultaneously. At each time tick we:

1. Maintain a matrix of activations for all neurons in all networks.
2. Identify which networks are still "live" (no output spike yet).
3. Apply the recurrent update to all live networks in one batched matrix multiplication and bias addition.
4. Clamp inputs, apply nonlinearities, and check for output spikes.
5. Record decisions for networks whose outputs fired and remove them from the live set.

This continues until all networks have either decided or reached $T_{\max}$. The approach yields large speed-ups by leveraging optimized linear algebra libraries and naturally records decision times, which can be incorporated into fitness to reward faster decisions.

V. ASSESSMENTS: SNAKE GAME EXPERIMENT

To test the model framework we chose the classic Snake game as an experimental environment.

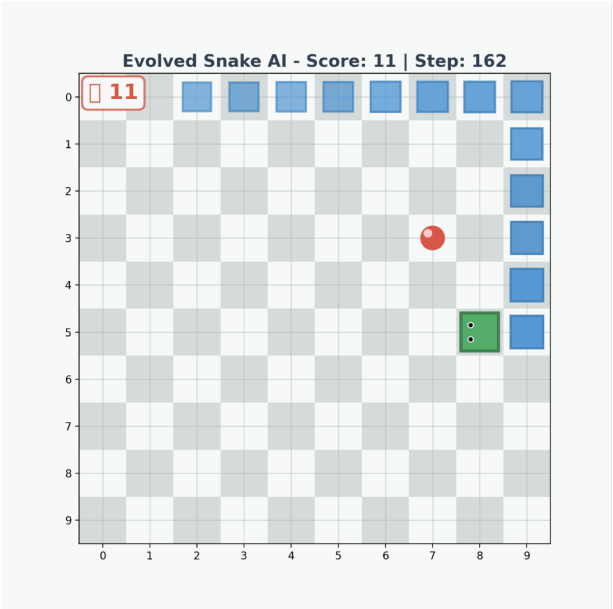

FIG. 6. Python visualization of the neuromorphic agent acting within the classic Snake environment, which serves as the testbed for the evolutionary spiking neural network experiments. The snake extends its length by consuming the red apple while avoiding self- and wall-collisions.

Our game is implemented on a 10×10 grid. The snake occupies a sequence of grid cells; at each time step it moves one cell in its current direction. The agent chooses actions relative to the current heading: 0 for keep straight, 1 for turn left, 2 for turn right. After the action is applied, the snake's head moves, the body follows, and collisions or food are checked. If the head moves off the grid or into its own body, the game ends. Food appears one piece at a time at random unoccupied positions; when eaten, the snake grows by one segment and a new food pellet is spawned. Episodes are limited to a maximum of 200 steps to discourage endless wandering.

A. State representation

Rather than feed the raw grid into the network, we use a compact, egocentric representation inspired by common Snake agents:

- Three binary danger sensors: `danger_straight`, `danger_left`, `danger_right`, indicating whether moving in that relative direction from the head would cause immediate collision with wall or body.
- Four one-hot direction indicators: `dir_up`, `dir_down`, `dir_left`, `dir_right`, encoding the current heading.
- Four binary food direction indicators: `food_up`, `food_down`, `food_left`, `food_right`, indicating whether the food lies above, below, left, or right of the head.

These eleven values form an input vector $\mathbf{x} \in \mathbb{R}^{11}$, all in $\{0, 1\}$ (cast to `float32`), and feed directly into the 11 input neurons. This compact representation gives the network exactly the information needed to reason about immediate danger and relative food position.

B. Evolutionary training procedure

We initialize a population of $N = 150$ networks, each with $n_{\text{in}} = 11$ input neurons, $n_{\text{hid}} = 15$ hidden neurons, and $n_{\text{out}} = 3$ output neurons. The connectivity mask allows all Input→Hidden, Hidden→Hidden (no self-loops), and Hidden→Output connections. Initial weights are drawn from a uniform distribution in $[-0.5, 0.5]$ and multiplied by the mask; biases are similarly initialized, with input biases set to zero so inputs pass through linearly.

Evaluation. To evaluate a network, we let it play 5 games of Snake in freshly initialized environments. During each game, at every time step we:

1. Encode the current state as the 11-dimensional vector $\mathbf{x}$.
2. Run the network's recurrent dynamics (with clamped inputs) until an output neuron spikes or a recurrent step limit (20 steps) is reached.
3. Choose the action corresponding to the first spiking output; if no output spikes, use the index of the maximal output as a default.
4. Apply the action to the environment and continue until the snake dies or the episode hits its step limit.

Rewards are assigned as follows: +10 for each food pellet eaten, -10 upon death (collision with wall or self), +0.1 for moves that reduce Manhattan distance to the food, -0.1 for moves that increase Manhattan distance to the food.

If the max step limit is reached without death, the game ends with no additional penalty or reward. The total reward for a game is the sum of these terms. The network's fitness is the average total reward across the 5 games; we also track average score (foods eaten) as a secondary metric.

Selection and reproduction. After evaluating all networks, we rank them by fitness and retain the top 10% as elites. These elites are carried unchanged into the next generation. The remainder of the population is filled by mutated copies of randomly chosen elites, using the following mutation scheme:

- *Weight mutation:* Each weight W_{ij} has a 15% chance to be perturbed by Gaussian noise drawn from $\mathcal{N}(0, 0.3)$ (respecting the connectivity mask).
- *Bias mutation:* Each bias B_j has a 15% chance to be perturbed by Gaussian noise $\mathcal{N}(0, 0.3)$.
- *Structural mutation:* With 15% probability per child, we attempt to add a new connection. We sample a pair of neurons (i, j) uniformly; if adding W_{ij} respects the connectivity rules and the mask currently disallows it, we enable the connection and initialize its weight from $\mathcal{N}(0, 0.2)$.

We run this evolutionary process for 500 generations. Throughout, we log best and mean fitness, best score per generation, and running best-ever score.

VI. RESULTS

We evolved a population of 150 Axia networks for 500 generations on the 10×10 Snake environment. Each individual was evaluated over five games per generation; fitness combined total reward (food, survival, and shaping terms) averaged over games. This section summarizes the quantitative learning dynamics and the qualitative behaviours that emerged.

A. Reward Dynamics Across Generations

Training produced a clear improvement in both the best and mean rewards over generations (Fig. 7). In the first few generations, most networks behave essentially at random: they frequently collide with walls or their own bodies within a handful of moves and rarely eat any food. As evolution proceeds, the best reward rises sharply during the first ~ 50 generations and then gradually levels off, while the population mean reward climbs more slowly but continues to increase throughout training.

To highlight coarse learning stages, we grouped generations into three phases: struggling (Gen 0–30), breakthrough (Gen 30–100), and refinement (Gen 100–500). Figure 8 shows the mean reward in each phase. Early in training the mean reward is low (≈ 5), reflecting short lifetimes and almost no food collection. During the breakthrough window the mean reward rises to around 30, indicating that a subset of individuals has discovered basic survival and food-chasing behaviours. In the refinement phase the mean reward climbs to over 100, corresponding to agents that routinely collect several food pellets while avoiding catastrophic self-collisions.

B. Score Progress and Final Performance

We next examine performance in terms of raw game score: the number of apples eaten per episode. Figure 9

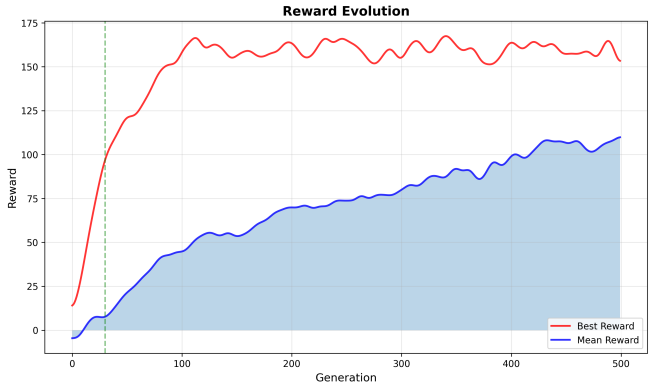

FIG. 7. Reward evolution over 500 generations with a population of 150 networks. The red curve shows the best reward in each generation; the blue curve shows the population mean reward. The dashed line marks the onset of consistent improvement as evolution leaves the purely random regime.

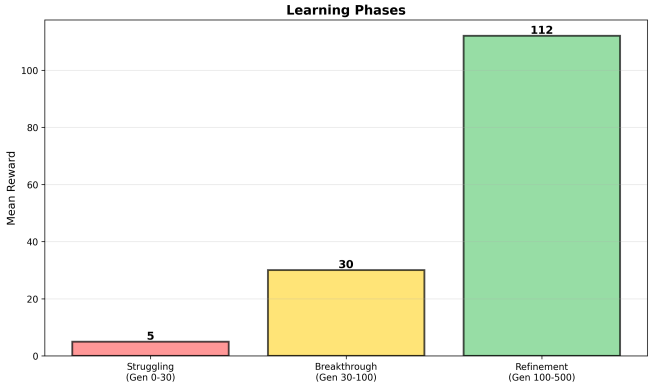

FIG. 8. Mean reward grouped into three evolutionary learning phases: struggling (Gen 0–30), breakthrough (Gen 30–100), and refinement (Gen 100–500). The bars summarize how quickly the population moves from near-random play to competent behaviour.

plots the best score in the population for each generation. The champion networks quickly progress from scores of 2–3 apples to a plateau around 14 apples by approximately generation 100. Occasional spikes in later generations correspond to especially successful individuals; the best-ever network discovered in our run achieves a score of 21 apples on a 10×10 grid.

To characterize the final population, we evaluated all individuals from the last generation on a fresh batch of games and plotted the distribution of mean scores (Fig. 10). The histogram is centered around a mean of approximately 11.4 apples, with a long right tail extending to the champion network at 21 apples. A random policy under the same conditions typically dies before reaching any food or, at best, collects a single apple, so the evolved population outperforms this baseline by an order of magnitude in expected score.

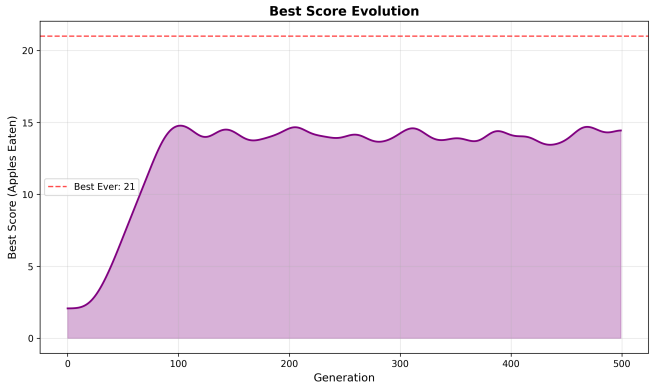

FIG. 9. Best score (apples eaten) per generation. Performance rises rapidly in the early generations and then oscillates around a plateau of roughly 14 apples, with a best-ever score of 21 marked by the dashed line.

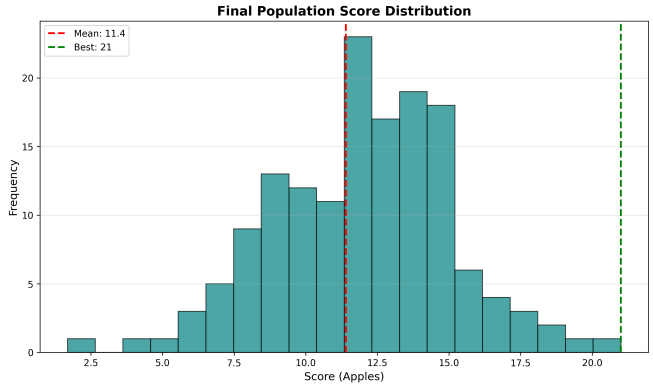

FIG. 10. Distribution of mean scores (apples eaten) for the final population. The red dashed line indicates the population mean ($\approx$ 11.4 apples) and the green dashed line marks the best network (21 apples).

C. Emergent Behaviour and Decision Dynamics

Qualitatively, the evolved agents display several consistent behavioural motifs, which were not hard-coded into the architecture:

Wall following.: Many successful lineages learn to trace the outer boundary of the grid when food is distant, reducing the risk of immediate self-intersection and keeping future manoeuvring room available.

Food homing.: When the food lies predominantly to one side of the snake, activity in specific hidden neurons drives the corresponding output neuron to fire, producing a turn toward the food as long as the local danger sensors remain inactive. This yields a robust "homing" behaviour that repeatedly steers the snake along efficient approach paths.

Self-avoidance.: As the snake grows longer, agents often prefer arcs that preserve open space ahead rather than closing tight loops. While this behaviour is not globally optimal, it markedly reduces the frequency of self-collisions compared to early generations.

In terms of decision speed, trained networks typically commit to an action within a few recurrent update steps, and in practice often within a single forward call because the 11-dimensional input is highly informative. Indecision events in which no output neuron spikes by the maximum time horizon become rare in late generations; when they do occur, they are resolved by selecting the output with the highest membrane potential. Overall, evolution discovers policies that are both *accurate* (high scores and rewards) and *decisive* (rapid spike-based action selection), supporting the viability of the Axia architecture for sequential decision-making tasks.

VII. DISCUSSION

The Snake experiments highlight both the strengths and limitations of this framework.

On the positive side, the results demonstrate that evolution of spiking, recurrent circuits can produce competent sequential decision policies without any gradient-based learning. Temporal dynamics and competition among outputs yield a natural mechanism for variable decision times and implicit confidence: clear-cut situations lead to rapid spikes; ambiguous states delay or prevent decisions. Structural evolution allows networks to add useful connections over generations, giving rise to richer dynamics and emergent heuristics like wall-following and self-avoidance.

However, several challenges remain. Evolution required hundreds of generations and thousands of game simulations to reach good performance, far less sample-efficient than reinforcement learning with backpropagation would likely be. Fitness is assigned at the episode level. Without intermediate gradient information, it is harder for evolution to fine-tune individual weights. Shaping rewards mitigate this but do not fully resolve the issue. The evolved networks are black boxes. While we can qualitatively characterize behaviour, it is difficult to attribute specific roles to specific neurons or connections without additional analysis tools. The networks are specialized to the particular Snake setup (grid size, reward structure). Generalizing to different layouts or rules would likely require further evolution or training under varied conditions.

Despite these limitations, the experiment serves as a proof of principle that population-level search over spiking recurrent circuits can lead to non-trivial, goal-directed behaviour in a sequential environment.

VIII. CONCLUSION AND FUTURE DIRECTIONS

We have built an evolutionary spiking neural network architecture built from first principles inspired by neuroscience. Instead of using backpropagation to adjust weights, the networks learn over generations, with populations mutating and competing based on performance. Neurons integrate inputs over time and fire discrete spikes; recurrence, leakage, and threshold competition collectively implement temporally extended computation and decision-making.

In the Snake environment, we showed that this approach can evolve an agent that navigates and eats food effectively. Starting from random behaviour, evolution discovers networks whose spiking dynamics encode useful heuristics and policies. This outcome is achieved without any gradient-based optimization, highlighting the potential of evolutionary methods to explore unconventional regions of network space.

Looking ahead, the most intriguing question is scalability. Snake is primarily a reactive control problem; chess poses a much deeper, strategic challenge. Evolving a chess-playing network from scratch would require substantial innovations, such as richer input encodings (piece type, colour, and position), action representations over a large variable legal move set, mechanisms for internal simulation of hypothetical moves using recurrent dynamics, curriculum strategies or modular evolution to progressively increase task complexity.

Hybrid approaches are particularly promising. One could use evolution to design network topology and initial weights, then fine-tune with gradient-based reinforcement learning, or conversely use gradient descent first and evolution later to improve robustness and discover structural variations. Another direction is developmental encoding: instead of genomes directly specifying every weight, they could encode growth rules for networks, reducing search dimensionality and potentially improving generalization.

Finally, any attempt to deploy evolutionary neural systems beyond toy domains must grapple with safety and interpretability. Evolution can exploit reward loopholes and produce unexpected strategies; careful reward design, constraints on growth, and post-hoc analysis tools will be essential.

This network is truly a complementary perspective on neural computation: learning by evolution rather than by backpropagation, with spiking, recurrent dynamics at its core. Our Snake experiments demonstrate feasibility and open a route toward more ambitious neuromorphic evolution in complex strategic environments.

[1] W. Maass, "Networks of spiking neurons: The third generation of neural network models," Neural Networks **10**, 1659–1671 (1997).

[2] F. Ponulak and A. Kaśinski, "Introduction to spiking neural networks: Information processing, learning and applications," Acta Neurobiologiae Experimentalis **71**, 409–433 (2011).

[3] D. Silver *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," Science **362**, 1140–1144 (2018).

[4] P. Dürr, C. Mattiussi, and D. Floreano, "Evolution of spiking neural networks for neuromorphic hardware," in *Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems* (IEEE, 2008), pp. 26–33.