Experience-Dependent Axonal Plasticity in Large-Scale Spiking Neural Network Simulations

Lars Niedermeier* and Jeffrey L. Krichmar†
*Niedermeier Consulting, Zurich, ZH, Switzerland
†Department of Cognitive Sciences, Department of Computer Science, University of California, Irvine, CA, USA
Correspondence Email: jkrichma@uci.edu

2023 International Joint Conference on Neural Networks (IJCNN) | 978-1-6654-8867-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/IJCNN54540.2023.10191241

***Abstract*—Axonal plasticity describes the biological phenomenon in which the myelin sheath thickness and the amplification of a signal change due to experience. Recent studies show this to be important for sequence learning and synchronization of temporal information. In spiking neural networks (SNNs), the time a spike travels from the presynaptic neuron along the axon until it reaches a postsynaptic neuron is an essential principle of how SNNs encode information. In simulators for large scale SNN models such as CARLsim, this time is modeled as synaptic delays with discrete values from one to several milliseconds. To simulate neural activity in large-scale SNNs efficiently, delays are transformed as indices to optimized structures that are built once before the simulation starts. As a consequence, and in contrast to synaptic weights, delays are not directly accessible as scalar data in the runtime memory. In the present paper, we introduce axonal delay learning rules in the SNN simulator CARLsim that can be updated during runtime. To demonstrate this feature, we implement the recent E-Prop learning rule in a recurrent SNN capable of flexible navigation. Compared to other studies for axonal plasticity that are based on LIF neurons, we also develop the SNN based on the more biologically realistic Izhikevich neural model. The present work serves as reference implementation for neuromorphic hardware that encode delays and serves as an interesting alternative to synaptic plasticity.**

***Index Terms*—Axonal Plasticity, Backpropagation Through Time (BPTT), Cognitive map, E-Prop, Hippocampus, Myelin Sheath, Navigation, Path Planning, Preplay, Simulation, Spiking Neural Network (SNN), Synchronization, Vicarious Trial and Error (VTE).**

I. Introduction

Experience-dependent axonal plasticity changes the conductance delays and has been shown to be important for learning skills and other cognitive behaviors [1]. Recently, a computational model of axonal plasticity was introduced that learned its environment, rapidly adapted to change and planned efficient routes to goals [2]. The model extended the spiking wavefront propagation path planner [3] by incorporating the E-Prop learning algorithm [4], that learns by adjusting the axonal delays, also called axonal plasticity, sketched in Fig. 1. The estimated time to travel between locations was captured in the axonal delays between neurons representing place. Thus, faster signal propagation reflected more efficient paths through the environment.

Axonal plasticity is fundamentally different compared to other SNN learning algorithms like STP or STDP, which alter synaptic weights (i.e., synaptic plasticity). To simulate neural activity in large-scale SNNs efficiently, delays are transformed into indices to optimized structures that are built once before the simulation starts. Simulators like CARLsim use sparse representations of synaptic connections in which they are internally ordered by their axonal delays [6].

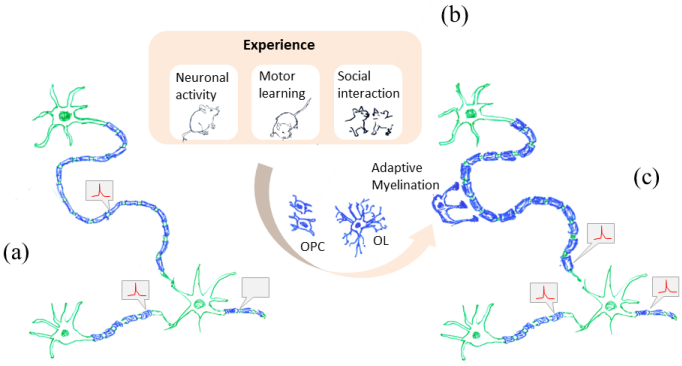


Fig. 1. **Axonal Plasticity.** Hypothetical circuit adopted from [1], [5]. The speed of spike transmission depends on the thickness of the myelin sheath and the number of signal amplifiers, called nodes of Ranvier. (a) Non-synchronous spike arrival as the myelin sheath is thin [1]. (b) Experience and neural activity cause axon alterations carried out by the glial cells oligodendrocyte precursors (OPCs) that differentiate into oligodendrocytes (OLs). (b) Thicker myelin sheath results faster conductance velocity, which can synchronize activity and causes the post-synaptic neuron to fire more reliably.

CARLsim has an intrinsic state model with transitions from CONFIG → SETUP → RUN [7]. Fig. 2 shows the methods applicable at each state.

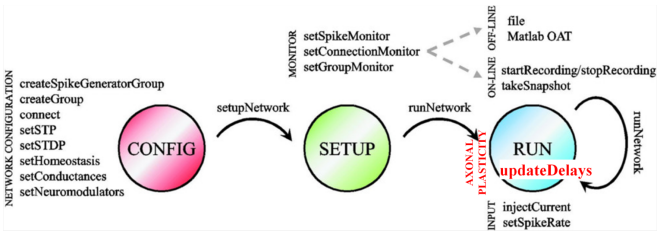


Fig. 2. The CARLsim state model extended with axonal plasticity functionality. Adapted from the CARLsim user guide [8], extensions in red font.

In the CONFIG state, the whole network is configured and located in host memory. The weight and the delay of each synaptic connection can be accessed directly. The following code snippet shows the C++11 iterator as it is used internally in $setupNetwork()$.

```
for (std::list<ConnectionInfo>::iterator
 connIt = postConnectionList.begin(); ...
   ...
   uint8_t  d = connIt->delay;
```

In order to prepare a network for execution, the CARLsim method $setupNetwork()$ compiles the network model to highly optimized runtime structures in backend memory, e.g., GPU device memory. In the resulting SETUP state, the explicit delays between the pre- and post-synaptic neurons are dissolved into optimized memory structures, for example $runtime[netId].postDelayInfo$.

The new method $updateDelays(\Delta D)$ therefore modifies the runtime structures mentioned above (e.g. $runtime[netId].postDelayInfo$) directly in back-end memory. Or in other words, $updateDelays$ can be imagined as a highly efficient incremental compile of the delta defined by sparse delay changes between pre- and postsynaptic neurons. Fig. 3 visualizes the induced requirement of maintaining the structural integrity at a conceptual level.

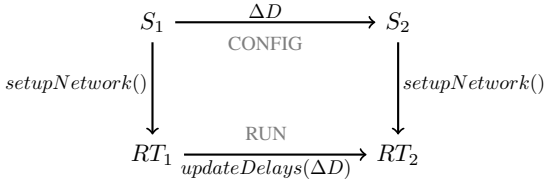


Fig. 3. Validation of the structural integrity for $updateDelays$. The runtime data structures $RT = (pre/postSynapticIds, postDelayInfo)$ are the result of $setupNetwork()$. S_1 is a SNN and ΔD defines the change of delays that transforms S_1 to the SNN S_2 in the state CONFIG. Applying $updateDelays$ in the state RUN with the same delay change ΔD results in the same runtime data structure.

In [2], a specific LIF neuron was adopted from [3] that used a delay counter with a discrete precision of 1ms to generate the spike given by (1).

$$d_{i,j}(t+1) = \begin{cases} D_{ij} & \text{if } v_j(t) \leq 1 \\ max(d_{ij}(t) - 1.0) & \text{otherwise} \end{cases} \quad (1)$$

The input current I at time $t+1$ is given by (2).

$$I_i(t+1) = \sum_{j=0}^{N} \begin{cases} 1 & \text{if } d_{ij}(t) = 1 \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

The input current drove the membrane potential v_i as shown in (3).

$$v_i(t+1) = u_i(t) + I_i(t+1) \quad (3)$$

The recovery variable u_i of the LIF neuron at time $t+1$ was given by (4).

$$u_i(t+1) = \begin{cases} -5 & \text{if } v_i(t) = 1 \\ min(u_i(t) + 1, 0) & \text{otherwise} \end{cases} \quad (4)$$

In contrast to [2] and [3], we use CARLsim to simulate more biologically realistic spiking neurons and allow axonal plasticity to be used in complex, large-scale networks. For instance, instead of discrete delay counter, CARLsim applies a generic kernel for generating spikes for several neuron models. The numerical integration to resolve the differential equations can be configured by a single line. The following code snippet shows how a simulation CARLsim can be configured using the Runge-Kutta numerical method with $\frac{1}{10}$ ms for the differential operator $\frac{d}{dt}$. [8]

```
sim.setIntegrationMethod(RUNGE_KUTTA4,10);
```

To demonstrate axonal plasticity on larger and complex SNNs we present a model that shows flexible path planning utilizing E-Prop learning and the Izhikevich neuron. The Izhikevich neuron model offers good trade-offs between biological accuracy and computational feasibility [9]. While CARLsim also supports LIF, this simple neuron model may not be adequate for biologically realistic networks. When it comes to integration of more sophisticated properties, such as neuromodulation or the integration of other neuron types like a resonator, for large-scale SNNs with high biological detail, the Izhikevich neuron model is a preferred choice.

In this paper, we advance the computational model for flexible navigation [2] as an extension for CARLsim, that enables large-scale SNNs. This work provides two major contributions:

- We extend the CARLsim kernel and interface by the method $updateDelay(..)$ that can update delays during simuation (RUN state), the method $findPath(..)$ that extracts the path from the spiking wavefront propagation including the calculation of the eligibility traces, and the an implementation of E-Prop, that orchestrates the methods above.
- We simulate the path planning and navigation results of [2], which used LIF neurons, with SNNs based on the Izhikevich neuron model. We can reproduce these results with similar performance and minimal impact to computation.

II. Methods

A. Spiking Neural Network Model

1) Izhikevich Neuron Model: We use the 4-parameter model of Izhikevich neurons that is described by the following equations [10].

$$\dot{v} = 0.04v^2 + 5v + 140 - u + I \quad (5)$$

$$\dot{u} = a(bv - u) \quad (6)$$

$$\text{if } v > 30 \begin{cases} v = c \\ u = u + d \end{cases} \quad (7)$$

The dot notation indicates the differential operator $\frac{d}{dt}$. The 4-parameter model is dimensionless and well suited for parameter tuning.

2) Place Cells: Place Cells were configured as regular spiking (RS) neurons with $a = 0.02, b = 0.2, c = -65, d = 8.0$ for the 4-parameter model. As we need 1 ms precision, we applied current-based (CUBA) synapses in CARLsim to the neuron group modeling the place cells.

$$I_i^{syn} = \sum_{i=1}^{N} s_{ij} w_{ij} \tag{8}$$

The weights between two place-cells, w_{ij}, were set to 85, so that the postsynaptic neuron fires once it receives a single presynaptic spike.

3) Interneurons: As the navigating agent can move from one location to another adjacent location, this must be reflected in the synaptic connections between place cells. For instance, Fig. 4b shows connectivity of an SNN simulating a $4x4$ environment. As a consequence, the network is recurrently connected and a forward excitement of the presynaptic neurons, produces direct and indirect cyclic excitements which would ultimately result in an exponential neural activity. Thus, when a place cell neuron fires, it needs to be inhibited to prevent the immediate feedback excitement from neighboring neurons.

Therefore, a layer of interneurons was added to the network with (1:1) recurrent connections, as shown in Fig. 4c. The design was inspired by the interneurons observed in the Hippocampus. If the place cell excites its neighbors, the inhibitory interneuron is excited slightly in the future. Then the interneuron inhibits the place cell neuron. The parameters for the inhibitory neurons were $a = 0.02, b = 0.2, c = -50, d = 2.75$, which is used to model fast spiking interneurons. The weights were set at a high value (i.e., 200) to temporarily silence the place cell even if all presynaptic neurons fired at the same time.

As the loopback could be delayed up to a certain time defined by the simulator (e.g., 20ms as default in CARLsim), the synaptic connections between the interneurons and place cells are conductance-based (COBA). In contrast to CUBA, see (8), for which the input current lasts for a single time step, in COBA, the input current decays exponentially over time. The ion channel of the synaptic receptors are described by the equations (9)-(12) (see [8]).

$$I_i^{syn} = \sum_{i=1}^{N} i_{ij} w_{ij} \tag{9}$$

The inhibitory interneurons are exited with each i_{ij} as the total of the current i_{AMPA} and the voltage dependent current i_{NMDA}.

$$i_{AMPA} = g_{AMPA}(v - v_{AMPA}^{rev}) \tag{10}$$

$$i_{NMDA} = g_{NMDA} \frac{[\frac{v+80}{60}]^2}{1 + [\frac{v+80}{60}]^2} (v - v_{NMDA}^{rev}) \tag{11}$$

f denotes the firing of a neuron as distinct spike event and t_f represents the time of this spike event. The Heaviside step function Θ select spike events in the past.

$$g_k = \sum_f e^{t - t_f/\tau} \Theta(t - t_f) \tag{12}$$

The following code snippet shows how the conductances is set interneurons and parameter are tuned so that the spiking wave front traverses the network without such unwanted echo effects at around 20 ms.

```
sim.setConductances(gInter, true,
 2, 25, 3, 25);//AMPA, NMDA, GABAa/b
```

4) Maze Environments and Traversal Costs: The environments in [2] were defined by cost matrices denoting the traversal cost between spatial locations. It is assumed that the agent can observe those environment costs while traversing through the maze. Fig. 4a shows the 4x4 example maze of [2] and Fig. 4b how it is mapped to a corresponding SNN with with 4x4 place cell neurons. The optimal path from the spatial locations defined in Cartesian coordinates from (1,1) to (4,4) is marked in red.

Fig. 4c shows the inhibitory neurons that are recurrently connected to each place cell neuron that enables the spiking wave propagation through the network. In the example maze and other mazes like in the Boone and Tolman experiments, the agent can move in four cardinal directions (North, East, South, West). Fig. 4d shows the resulting bidirectional connections in black for a minimalistic 3x3 network. In other environments, like the Morris water maze, the agent also can move diagonal (e.g., Northeast) which is marked in blue in Fig. 4d and resulting in eight cardinal directions for the neuron.

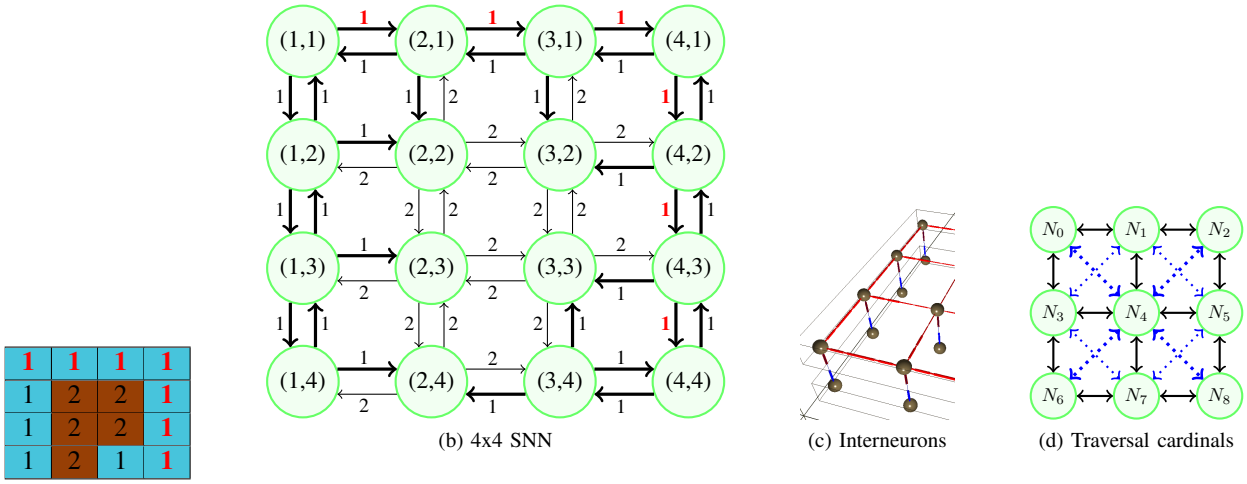
B. Spiking Wavefront Propagation and Path Extraction

Fig. 5a shows the propagation of the spiking wavefront in an SNN with $1,024$ place cells. It maps an open field maze with the dimension 32x32 with costs as random variable of the discrete uniform distribution $\mathcal{U}\{1, 7\}$ ms. During path planning, the starting place cell (S) near the center of the bottom right quadrant was excited. The spiking expands like a wave until it hits the the target place cell (E). Fig. 5b shows the resulting path in magenta. The environment costs $map(x, y)$ of the maze are in Cartesian coordinates and presented as heat plot.

As the neural activity in the place cells was well controlled by the interneurons (see II-A3), the spiking wavefront propagated over time and each neuron yielded an eligibility trace as defined by the following equation.

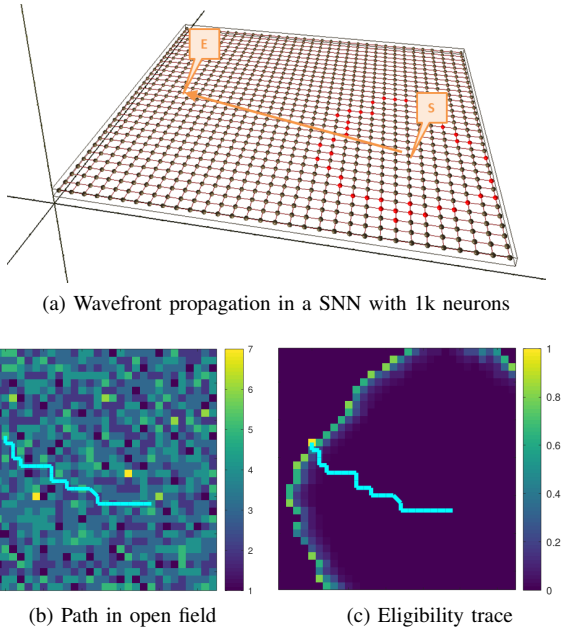
$$e_i(t+1) = \begin{cases} 1 & \text{if } v_i(t) \geq 1 \\ e_i(t) - \frac{e_i(t)}{\tau} & \text{otherwise} \end{cases} \tag{13}$$

Fig. 5c shows the resulting eligibility trace as heat map and with the path as reference. Interestingly, the eligibility trace can be intuitively associated with a wave traveling through the maze with decreasing strength into the past. This is the basis for the E-Prop learning rule. The amount of training loops and errors induced by time constant τ in (13) was



(a) 4x4 maze (b) 4x4 SNN (c) Interneurons (d) Traversal cardinals

Fig. 4. Mapping an environment to an SNN. (a) 4x4 maze with traversal costs leading to an optimal path from (1,1) to (4,4). (b) Corresponding SNN with 4x4 place cells and their corresponding connection delays. (c) Interneurons of the 4x4 network. They are recurrently connected to place cells. Blue marks the inhibitory synaptic connection. (d) A minimal 3x3 SNN to show the synaptic connections in a maze with four (black) and eight agent movement directions (black+blue).



(a) Wavefront propagation in a SNN with 1k neurons

(b) Path in open field (c) Eligibility trace

Fig. 5. Spiking network with 1024 neurons for an open field environment with edge length 32 and costs of a uniform distribution $\mathcal{U}\{1, 7\}$. The delays of the spiking network map the corresponding traversal costs in ms. (a) Propagation of spiking wavefront in the SNN after the start neuron (S) was excited. Spiking neurons are marked red to visualize the discharge of the electrical action potential. The wavefront propagates through the network until it hits the end neuron (E). (b) Extracted path from S to E is shown in magenta. The traversal cost of each location of the maze is shown in a heat map with the range from 1 to 7 (which corresponds to the delays). (c) Eligibility trace induced by the propagating wavefront. Its values decay back over time from 1 to 0, see eqn. (13), and are colored according to the heat map.

investigated in [2] and fixed to the value of 25 for minimal errors. Consequently we use the same value to reproduce those results.

Fig. 5a shows the neural activity of the place cells, that propagates as a wave with the starting point as its center. The following code snippet shows how the extended interface of CARLsim is used to extract the path after the wavefront has hit the end point.

```
std::vector<int> path;
std::vector<float> eligibility;
sim.findWavefrontPath(path, eligibility,
  netId, gPC, startNId, endNId);
```

As the path can be of variable length it is written into a C++11 vector that holds the neuron indices of the place cells in corresponding order. The start and end points are passed as neuron ids ($startNId$, $endNId$) and the neuron group with the place cells is identified by gPC. For efficiency reasons, the method also covers the calculation of the eligibility trace. As defined in equation (13), each place cell is assigned a float value $\in [0, 1]$. The resulting eligibility traces are presented as heat map plots as shown in the Fig. 5c example and in later experiments.

C. E-Prop and BPTT

The eligibility trace of a place cell neuron provides the basis for the E-Prop learning rule, which is defined by the following equation.

$$D_{ij}(t+1) = D_{ij} + \delta(e_i(t))(map_{xy} - D_{ij}(t)) \qquad (14)$$

The delays D_{ij} encode the traversal costs of the environment as it was last observed by the agent. The difference of the actual traversal cost defined by the maze as $map(x, y)$ and the last observed cost defines the loss function in (14). The eligibility traces $e_i(t)$ encodes the temporal aspect of when the spike in the wavefront occurred, which is applied to the loss function in equation (14). In this way, E-Prop is a form of

Back-Propagation-Through-Time (BPTT). The learning rate δ is constant value set to 0.5. It is assumed, that the agent can observe the features of the environment along the path taken [2].

E-Prop updates the delays along the path and thus resolves the credit assignment problem. The eligibility trace $e_i(t)$ is applied for each neighbor j of the place cells neuron i that is contained in the path. For instance, if N_1 in Fig. 4d was contained in the path and the agent can move in four cardinal directions, it has three neighbors resulting in six updates in total. The following code snippet shows the E-Prop algorithm defined by equation (14). The eligibility trace was calculated when extracting the path. E-Prop recalculates D_i and then calls $updateDelays()$ to write them back in runtime memory.

```
std::vector<std::tuple<int,int,uint8_t>>
 connDelays; //container for new delays
for(auto iter = ... //all connections
 int pre = std::get<0>(*iter);
 int post = std::get<1>(*iter);
 uint8_t cost = maze->map.find(std::tuple
  <int, int>(pre, post))->second;
 float e_i = eligibility[post];
 uint8_t d_old = std::get<2>(*iter);//D(t)
 if(alongPath) { //D(t+1)
  uint8_t d_new = d + delta * e_i *
   (cost - d); //loss
   connDelays.push_back(std::tuple
    <int,int,uint8_t>(pre, post, d_new));
...
sim.updateDelays(gPC, gPC, connDelays);
```

D. updateDelays

Sparse representation of synaptic connections enable simulation of large-scale SNNs as it reduces the required memory from $\mathcal{O}(NDM)$ to $\mathcal{O}(N(M+D))$, where N is the number of neurons, M the number of synapses, and D the maximal axonal delay [6]. Fig. 6a illustrates the sparse representation utilizing a minimalistic SNN with four neurons and three synaptic connections. The algorithm of $updateDelays$ modifies arbitrary delays (denoted as ΔD) in runtime memory and is presented in Fig. 6b. The other parameters of the method identify the context by supplying the network partition ($netId$), and the pre-/ postsynaptic neuron groups ($gGrpIdPre$, $gGrpIdPre$). Fig. 6c (6d) shows how a change from 4 ms to 2 ms (1 ms) of the synaptic connection between neuron 0 and 2 affects the data structures in runtime memory.

III. Results

A. Structural Integrity of Runtime Data

Unit tests were developed for minimal SNNs up to full scale SNNs with random generated delays and changes to validate that $updateDelays$ keeps the structural integrity intact. As outlined above, CARLsim uses sparse representation of synaptic connections. The method $setup()$, called in state CONFIG, translates the delays of the SNN model to the runtime data structures $pre/postSynapticIds$ and $postDelayInfo$. Fig. 3 outlines the principle of the validation of structural integrity: When the SNN is changed by $updateDelays()$ in the state RUN, the resulting runtime structures are the same as if the delays are changed in CONFIG state first and then translated by $setupNetwork$.

Let S_1 be an SNN with n neurons N_i and $i \in 1..n$. Its synaptic connections between N_i, N_j, $(i,j) \subseteq (1..n, 1..n)$ have delays of D_{ij} ms. ΔD_{kl} defines the change of delays as subset $(k,l) \subset (i,j)$ of those connections. RtD_2 results by applying $updateDelays(\Delta D_{kl})$ in the state RUN of $S1$. It has to be equal to $setupNetwork$ on S_2 which has the same N_i but altered synaptic connections ΔD_{kl} in the state CONFIG.

While this procedure validates the static structure of the network, its dynamics were investigated with concrete applications for axonal plasticity as presented next.

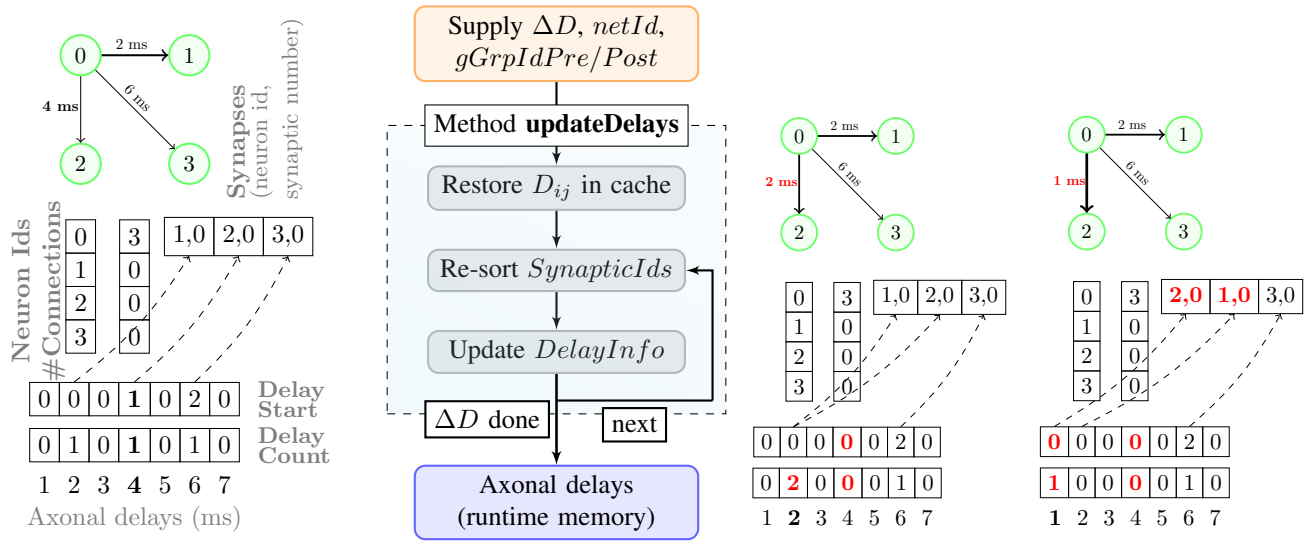
B. Validation by Flexible Path Planning Applications

The functionality of the SNN model was validated by replicating human and rodent navigation experiments in the mazes described in [2]. The environment costs were mapped to the delays of the SNN. The VTE activity, spiking wave propagation, and paths in the present work were similar to those reported in [2]. $updatedDelays$ was called from E-Prop (BPTT) during runtime of the simulation and without interruptions.

1) Human navigation studies by Boone: Fig. 7a shows the virtual environment used in human navigation studies [11]. It can be seen as a maze in which each location has an (x,y) coordinate and is assigned a traversal cost as shown on the heat map. The borders of the maze (yellow color) and non-traversalable areas (green color) are mapped to high costs (equivalent to a 5 ms delay), while the traversal areas (purple color) are mapped to the minimal cost (equivalent to a 1 ms delay).

As in the human study, agents were learned a fixed route over multiple training loops. Route sections during training are marked in cyan in Fig. 7a. Each route segment has start point S=(x,y) and end point E=(x,y) in Cartesian coordinates of the maze. For instance, in the example above the route segment starts at S=(12,12) and ends at E=(12,6). All delays were initialized with high values that were $\leq maxDelay$, here 7 ms. The agent preplays the possible paths with a spiking wave through the place cells, e.g. starting at S=(12,12). When the spiking wavefront hits the target neuron, e.g. E=(12,6), the eligibility traces resulting from path extraction, are applied by the E-Prop learning algorithm. An eligibility trace resembles BPTT as it cools down from $1.0 \rightarrow 0.0$ over time to the values displayed in the heat map. Fig. 7b - 7d shows how the agent learns optimal paths through the maze.

After the training was complete, the navigation task was to find a path from a randomly chosen start and end points, which were located on the route, but not necessarily the shortest way to reach the goal. As in the human study [11] and the modeling study [2], the agent sometimes followed the trained routes, while other times it took novel shortcuts (see Fig. 7e and 7f).



(a) Sparse representation (b) Algorithm of $updateDelay$ (c) $\Delta D = 2$ ms (d) $\Delta D = 1$ ms

Fig. 6. Sparse representation of synaptic connections demonstrated in a minimalistic SNN. The algorithm of $updateDelay$ is applied for two essential cases and the resulting modifications in the runtime memory structures are marked in red color. (a) Sparse representation of synaptic connections dissolves delays into indices to data structures in runtime memory. The first neuron with id 0 of the minimalistic SNN has three connections to its post-synaptic neurons 1, 2, and 3. In sparse representation, synapses are ordered ascending by their axonal delays. For instance, the synapse for delay $D_{0,2} = 4$ms is in the middle between those for $D_{0,1}$ and $D_{0,3}$. It is encoded encoded in the forth entry of the axonal delays table. Delay Start references the position of the synapse. Delay Count encodes how many post-synaptic connections the neuron has. (b) Algorithm of the method $updateDelay$. (c) $\Delta D_{0,2} = 2$ ms increments the Delay Count at 2ms. The former entry Delay Start is set to 0 as this no longer references the second synapse. (d) $\Delta D_{0,2} = 1$ ms induces a re-sort of the synaptic connections by swapping the first two synapses. Delay Start becomes 0 and references the first synapse.

Fig. 7g shows how neural activity and loss decreases while the agent explores and learns the maze.

2) Rodent navigation in the Tolman detour task: In the Tolman detour task the navigation, the rat must adapt to find the appropriate detour when a learned path is blocked [12]. We reproduced similar results to [2] though with a slightly altered experimental setup due to address the default $maxDelay = 20$ms of CARLsim. The delays in the network were initialized with a random uniform distribution $\mathcal{U}\{2, 3\}$ and the absolute lengths of the detour were slightly reduced.

Fig. 8a shows a training trial for learning a straight path from the start to the goal. After the training, barrier P1 or P2 was placed in the middle corridor and the agent had to learn a detour to the goal. When the barrier P1 was set at the location (9,7), the agent discovered the new path on the 7th trial as shown in Fig. 8b. The high number of eligible traces (neural activity) suggests a great deal of uncertainty that corresponds with an early trial. When the barrier P2 was set at the location (6,7), the agent learned the new path on the 5th trial as shown in Fig. 8c. Both detours were learned loss free in less than ten trials. Fig. 8d shows the convergence of loss and neural activity while the agent learned the detour after barrier P1 was set.

3) Rodent Navigation in the Morris Water Maze: The Morris water maze is a standard test of hippocampal dependent spatial learning in rodents [13]. A rodent has to find a platform which is hidden beneath the surface of a circular area covered with opaque water. To ensure the navigation is based on distal cues instead of learned sequences of movements, each trial starts from a random location.

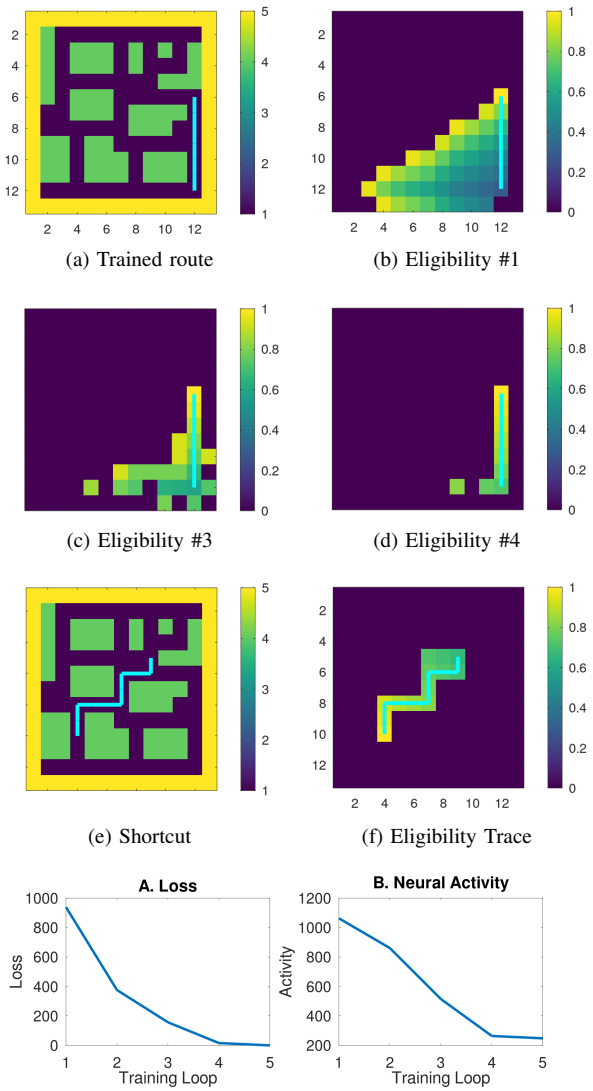
The Morris water maze was simulated by a 13x13 SNN which its neurons corresponding to the grid locations of the costs map. The platform was located at (5,5) and had the cost of 1. The cost of the area covered with water were assigned a random value with discrete uniform distribution $\mathcal{U}\{2, 5\}$. All other locations are not traversal and therefore were initialized with 120. The maze resembles an open field as the agent can navigate to eight cardinals points (N, NE, E, SE, S, SW, W, NW). For each of the 32 trials, the starting points were randomly chosen from the wall at one of the maze's four quadrants.

Fig. 9 shows the paths in cyan as the agent navigated from a randomly chosen start point on the east wall to the hidden platform. The path became visually less circuitousness in late trials and the number of eligibility trace respectively loss decreased as the agent learned the traversal costs by distal cues from former trials.

C. Performance

1) Performance Spiking Wavefront: The processing time for the propagation of the spiking wavefront depends on the utilized hardware[1], on the complexity of the SNN, and on the computational costs of the neuron model. In the present examples, the complexity is driven by the amount of place cells required to map the traversal costs of the environment, see Fig. 10a. The experimental setup defines the edge length L

[1]GPU: NVIDIA Titan Xp, CPU: Intel Xeon E5-2667 v4 3.2 GHz



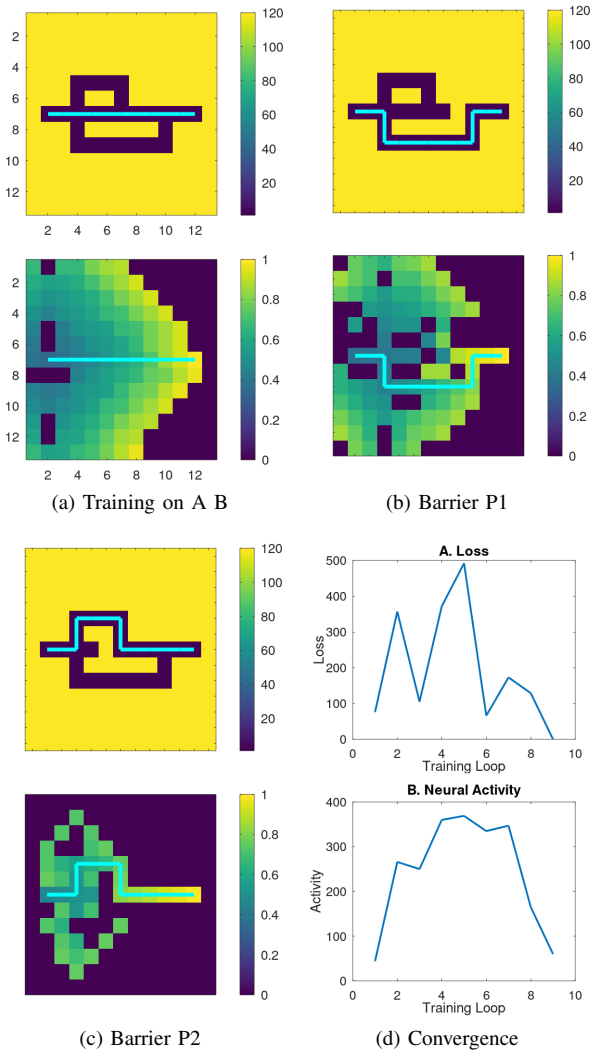
(a) Trained route (b) Eligibility #1

(c) Eligibility #3 (d) Eligibility #4

(e) Shortcut (f) Eligibility Trace

(g) Convergence for learning the maze

Fig. 7. Training of fixed route segments in the Boone maze. (a) Trained route segment. (b) - (d) Eligibility traces of training session #1 to #4. (e) Found shortcut between two landmarks during trial. (f)Eligibility trace of shortcut prove learned features of the maze. (g) Loss and neural activity decrease indicates learning of the maze features.



(a) Training on A B (b) Barrier P1

(c) Barrier P2 (d) Convergence

Fig. 8. Tolman detour task to simulate rodent navigation from A=(2,7) to B=(12,7). The resulting path is displayed the maze in cyan color (upper row) with the corresponding eligibility trace below. (a) Agent was trained on to go from A=(2,7) to B=(12,7). (b) Agent discovered new path after barrier P1 was set at (9,7) at the 7th trial. (c) Agent discovered new path after barrier P2 was set at (6,7) at the 5th trial. (d) Convergence of loss and neural activity while learning the detour for barrier P1.

of the maze and the degree of freedoms the agent can move, in this case, four or eight cardinal points C. For large mazes, the order of the synaptic connections is $\mathcal{O}(C * L^2)$. For instance, a maze with an edge length $L = 32$ has 1k neurons and 8k connections.

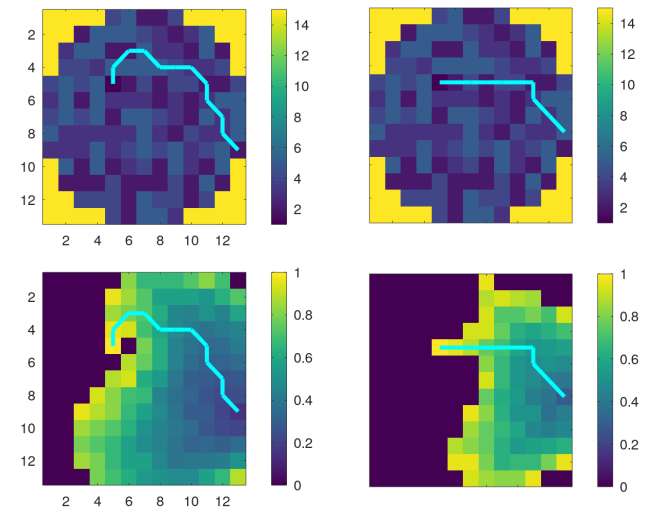
To measure the performance of the wavefront propagation, we activated the place cell at the top-left of each maze used in the experiments $S = (1, 1)$ and measured the processing time when the bottom-right $E = (L, L)$ was reached, e.g. S=(1,1) and E=(13,13) for Boone, Tolman, and Morris mazes.

Fig. 10b shows that Izhikevich RS neurons do have the equivalent performance as LIF for large models on GPUs in CARLsim. For small models with several hundreds of neurons, simulations on CPU run at the same speed or even faster. This corresponds to former performance analysis and other GPU based simulators. The integration method was set to Runge-Kutta4 with 10 steps ($dt = 0.1$ms) that is recommended for features for simulations with high biological detail such as multi-compartment compartments neurons [8].

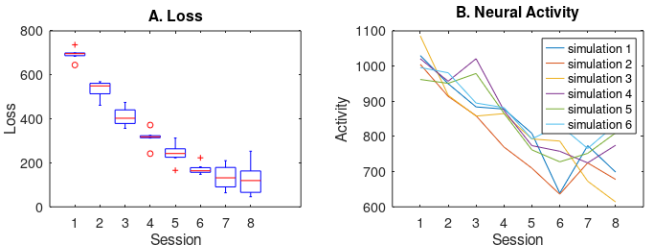
2) Performance UpdateDelays: In the problem domain, the majority of changes are sparse. For instance, changes are only applied along the path [2]. Also other connections outside the group like the interneurons needs to be considered. Fig. 11 shows the performance of the method $updateDelays$. which meets the design goal, that it has no performance impact.

IV. Discussion

We have introduced an implementation for axonal plasticity for large scale SNN simulations based on the Izhikevich

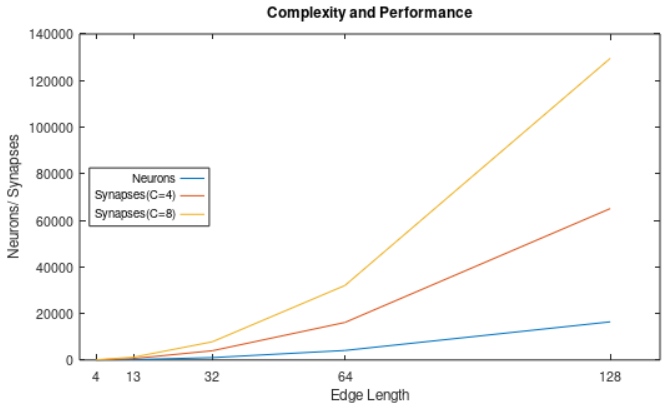


(a) Early trial (b) Late trial

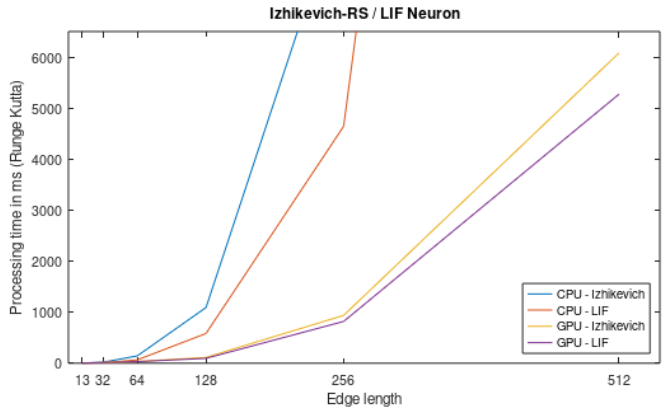


(c) Statistical convergence of loss and neural activity

Fig. 9. Morris Water Maze with random start points from the east border. (a) Early trial with the agent searching for the platform. (b) Straighter path of late trial due spatial learning. (c) Convergence of loss and neural activity in six simulations with eight sessions each.



(a) Complexity



(b) Processing time

Fig. 10. The processing time for the spiking wavefront propagation depends on the complexity of the SNN and the neuron model. (a) Complexity increases quadratically both for neurons and synapses in relation to the edge length of the maze. (b) Izhikevich neurons have an equivalent performance as the LIF neurons for large models on GPUs in CARLsim. CPUs are do perform well for small models. The integration method was set to Runge-Kutta4 with 10 steps ($dt = 0.1$ms).

neuron model. Although the present results used E-Prop as its learning rule [4], the implementation is flexible and would allow other learning rules, such as STDP, to be used to dynamically update axonal delays during runtime. The key was the development of the $updateDelays$ method, that enables CARLsim to update delays during runtime without interrupting the running simulation. This allows for efficient computation of large scale SNNs that support biological complexity.

Although the spiking wavefront path planning algorithm was inspired by recent findings in hippocampal replay [2], [3], the original and present work is not suggesting that axonal plasticity, such as that described by [1], is occurring during adaptive path planning. Axonal plasticity is a slower process than synaptic LTP and LTD and is probably more prominent for developmental learning and learning on longer timescales. However, axonal plasticity was introduced for SNNs to explore this novel form of neuronal plasticity that may have advantages for time-based coding algorithms and explainability (i.e, the learning can readily be read out in the delays). By making it available in CARLsim, we hope that future users will consider using this form of plasticity in their simulations.

One primary design goal was to avoid bottlenecks induced by the delay update for axonal plasticity. We have achieved this by strictly honoring the sparse representation of synaptic connections and its intrinsic runtime data structures that avoid searching by direct indexing. We expect the delay changes induced by E-Prop to be sparse in the present examples, as they only target those delays along the path. Furthermore, according to the requirements we are aware of, we estimated the time intervals to be similar to other learning rules like STDP or STP which is several tens to hundreds of ms. We have measured the performance for the known cases and pending applications and met the expected results. If future applications have different requirements, there are still feasible methods to optimize the performance, for instance by tuning the cache used in the first step of $updateDelays$.

Compared to prior work, which utilized a specialized LIF neuron model [2], [3], we have achieved a higher biological plausibility as we used Izhikivich neuron types in the present work. For instance, our place cells were simulated with regular spiking neurons since those are typically recorded in pyramidal

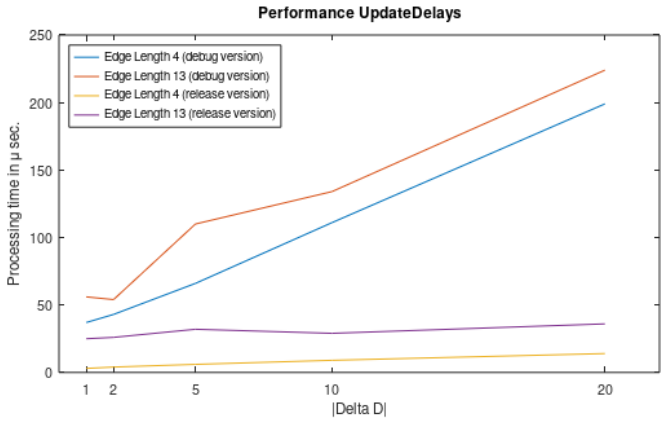


Fig. 11. The processing time of method $updateDelays$ for ΔD of increasing sizes and SNNs with edge lengths 4 and 13. As the implementation of the method is highly optimized, we have provided the results for the Debug version for better comparison.

cells found in the hippocmapus. The inhibitory interneurons neurons, were fast spiking similar to those observed in hippocampal area CA3 [14]. New features of CARLsim version 6, which allow the configuration of COBA and CUBA synapses at the group level, were necessary to reproduce the results in the present examples. We only tuned parameters manually and at a coarse level, to avoid the risk of overfitting. This minimalistic approach for the neural microcircuit was deliberately chosen to show a functional equivalent to the applications based on LIF of former work.

V. Conclusion

Axonal plasticity appears to be important for synchronization and skill learning [1]. It can change due to practice and experience. It can also be impacted due to neurological disorders. Despite its importance in brain function, it is rarely included in neural network simulations. The present work provides an efficient implementation of experience-dependent axonal plasticity. In future work, we will explore other learning rules that affect conductance delays and develop other applications.

VI. Acknowledgment

References

[1] R. D. Fields, "A new mechanism of nervous system plasticity: activity-dependent myelination," *Nature Reviews Neuroscience*, vol. 16, pp. 756–767, 2015.

[2] J. L. Krichmar, N. A. Ketz, P. K. Pilly, and A. Soltoggio, "Flexible path planning in a spiking model of replay and vicarious trial and error," in *From Animals to Animats 16*. Berlin, Heidelberg: Springer-Verlag, 2022, pp. 177 – 189. [Online]. Available: https://doi.org/10.1007/978-3-031-16770-6_15

[3] T. Hwu, A. Wang, N. Oros, and J. L. Krichmar, "Adaptive robot path planning using a spiking neuron algorithm with axonal delays," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 10, pp. 126–137, 2018.

[4] G. Bellec, F. Scherr, A. Subramoney, E. Hajek, D. Salaj, R. A. Legenstein, and W. Maass, "A solution to the learning dilemma for recurrent networks of spiking neurons," *Nature Communications*, vol. 11, 2019.

[5] C. W. Mount and M. Monje, "Wrapped to adapt: Experience-dependent myelination," *Neuron*, vol. 95, pp. 743–756, 2017.

[6] J. M. Nageswaran, N. Dutt, J. L. Krichmar, A. Nicolau, and A. Veidenbaum, "Efficient simulation of large-scale spiking neural networks using cuda graphics processors," in *2009 International Joint Conference on Neural Networks*, June 2009, pp. 2145–2152.

[7] L. Niedermeier, K. Chen, J. Xing, A. Das, J. Kopsick, E. Scott, N. Sutton, K. Weber, N. Dutt, and J. L. Krichmar, "Carlsim 6: An open source library for large-scale, biologically detailed spiking neural network simulation," in *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2022, pp. 1–10.

[8] UCI CARLsim Team, *CARLsim 6 User Guide*, Cognitive Anteater Robotics Laboratory, University of Calfornia, Irvine. [Online]. Available: https://uci-carl.github.io/CARLsim6

[9] K. Yamazaki, V.-K. Vo-Ho, D. Bulsara, and N. Le, "Spiking neural networks and their applications: A review," *Brain Sciences*, vol. 12, no. 7, 2022. [Online]. Available: https://www.mdpi.com/2076-3425/12/7/863

[10] E. M. Izhikevich, "Which model to use for cortical spiking neurons?" *IEEE Trans. Neural Netw.*, vol. 15, no. 5, pp. 1063–1070, Sep. 2004.

[11] A. P. Boone, B. Maghen, and M. Hegarty, "Instructions matter: Individual differences in navigation strategy and ability," *Memory & Cognition*, pp. 1–14, 2019.

[12] A. Alvernhe, E. Save, and B. Poucet, "Local remapping of place cell firing in the tolman detour task," *Eur J Neurosci*, vol. 33, no. 9, pp. 1696–705, 2011.

[13] R. G. M. Morris, "Developments of a water-maze procedure for studying spatial learning in the rat," *Journal of Neuroscience Methods*, vol. 11, pp. 47–60, 1984.

[14] J. D. Kopsick, C. Tecuatl, K. Moradi, S. M. Attili, H. J. Kashyap, J. Xing, K. Chen, J. L. Krichmar, and G. A. Ascoli, "Robust resting-state dynamics in a large-scale spiking neural network model of area ca3 in the mouse hippocampus," *Cognitive Computation*, 2022.