

SNS-Toolbox: A Tool for Efficient Simulation of Synthetic Nervous Systems^{*}

William R. P. Nourse¹[0000–0002–1437–026X], Nicholas S. Szczecinski²[0000–0002–6453–6475], and Roger D. Quinn³[0000–0002–8504–7160]

¹ Department of Electrical, Computer, and Systems Engineering,
Case Western Reserve University, Cleveland OH 44106, USA
`nourse@case.edu`

² Department of Mechanical and Aerospace Engineering,
West Virginia University, Morgantown WV 26506, USA

³ Department of Mechanical and Aerospace Engineering,
Case Western Reserve University, Cleveland OH 44106, USA

Abstract. We introduce SNS-Toolbox, a Python software package for the design and simulation of networks of conductance-based neurons and synapses, also called Synthetic Nervous Systems (SNS). SNS-Toolbox implements non-spiking and spiking neurons in multiple software backends, and is capable of simulating networks with thousands of neurons in real-time. We benchmark the toolbox simulation speed across multiple network sizes, characterize upper limits on network size in various scenarios, and showcase the design of a two-layer convolutional network inspired by circuits within the *Drosophila melanogaster* optic lobe. SNS-Toolbox, as well as the code to generate all of the figures in this work, is located at <https://github.com/wnourse05/SNS-Toolbox>.

Keywords: Conductance based modeling · Synthetic Nervous Systems · Simulation · Neurorobotics · Neural Networks.

1 Introduction

In recent years, more and more research has been done on implementing control systems for robots using networks of biologically-inspired neurons [6, 11, 14, 20] with an end goal of creating robots with the adaptability and generalization of animals. One particular approach in this field is using Synthetic Nervous Systems (SNS) [21], which are networks of conductance-based neurons and synapses connected using analytic design rules. The predominant tool used for designing SNS controllers until now has been Animatlab [5], a visual tool that combines neural simulation, physics simulation, and plotting within one software platform. Robot controllers have been successfully developed using this software [11, 14]. However, as the body of neuroscience knowledge increases and control networks

^{*} This work was funded by National Science Foundation (NSF) Award #1704436, as well as by NSF DBI 2015317 as part of the NSF/CIHR/DFG/FRQ/UKRI-MRC Next Generation Networks for Neuroscience Program

become larger, it is becoming more difficult to maintain and develop these large networks and simulate them in a sufficiently fast manner [19].

Simulators capable of designing conductance-based neurons have existed for years, such as NEURON [13], NEST [15], and Brian [15]. There are also simulators capable of designing large neural networks using principles of machine learning, albeit using reduced neural models, such as *snnTorch* [8], *SpykeTorch* [17], and *BindsNET* [15]. Although many simulators are available, it is difficult to interface any of these programs with hardware. Many of these were designed around collecting data over long simulation runs, so interfacing with hardware becomes an exercise in creative engineering [20]. There are solutions for robotic interfacing in simulation [9], but due to the overhead associated with these programs it is not practical to execute the network in real-time. High-performance robots have been developed which are controlled using spiking neurons [6], but these rely on specialized neuromorphic hardware and software specially designed for these platforms, which are not widely available [1]. As of now the best software for implementing conductance-based networks and applying them to real hardware is *Nengo* [15], but this software is primarily designed for networks using the Neural Engineering Framework [7] and contains enough overhead to slow down performance without the use of specialized hardware.

We present *SNS-Toolbox*, a Python software package for designing and simulating networks of conductance-based neurons and synapses. External inputs are processed and transformed into neural state outputs on a timestep-by-timestep basis. These outputs can then be used with any software or external hardware which is capable of communicating with Python code. By not interfacing with a dedicated physics simulator and optimizing for two specific neural models, we can simulate networks of thousands of neurons and synapses in real-time or faster using consumer-grade computer hardware.

2 Neural Models

When designing *SNS-Toolbox*, we chose to focus on designing and simulating two neural models that have often been used in previous SNS work: a non-spiking model and a spiking model.

2.1 Non-spiking Neurons and Synapses

Non-spiking neurons are simulated as leaky integrators, the same model as those used in continuous-time recurrent neural networks [2, 21]. The membrane depolarization above rest, U , behaves according to the differential equation

$$C_{mem} \frac{dU}{dt} = -G_{mem} \cdot U + I_{syn} + I_{bias} + I_{app}, \quad (1)$$

where C_{mem} is the membrane capacitance, G_{mem} is the membrane leak conductance, I_{bias} is a constant offset current, and I_{app} is an external applied current.

I_{syn} is the current induced by incoming conductance-based synapses to the neuron,

$$I_{syn} = \sum_{i=1}^n G_{syn}^i \cdot (E_{syn}^i - U). \quad (2)$$

E_{syn}^i is the reversal potential of the i th incoming synapse relative to the neuron's rest potential, and G_{syn}^i is the instantaneous synaptic conductance. This conductance is defined as a function of the pre-synaptic neuron depolarization,

$$G_{syn}^i = \max \left(0, \min \left(G_{max,non}^i \cdot \frac{U_{pre}}{R}, G_{max,non}^i \right) \right) \quad (3)$$

$G_{max,non}^i$ is the maximum possible conductance for the synapse, and R is the maximum desired membrane depolarization of any neuron throughout the network [21]. In general, $G_{max,non}^i$ controls the strength of a synapse while E_{syn} determines the behavior (excitatory, inhibitory, or modulatory) [21]. Substituting Equation 2 into Equation 1, the full non-spiking neural model can be written as

$$C_{mem} \frac{dU}{dt} = -G_{mem} \cdot U + \sum_{i=1}^n G_{syn}^i \cdot (E_{syn}^i - U) + I_{bias} + I_{app}. \quad (4)$$

2.2 Spiking Neurons and Synapses

For the spiking model, the membrane depolarization dynamics are similar to the non-spiking model (see Equation 4), with an additional dynamical variable for a firing threshold θ [22],

$$\tau_\theta \frac{d\theta}{dt} = -\theta + \theta_0 + m \cdot U, \quad (5)$$

where τ_θ is the threshold time constant, θ_0 is the initial threshold voltage, and m is a proportionality constant describing how changes in U affect θ . We also define a spiking variable δ , which represents a spike and resets the membrane state,

$$\delta = \begin{cases} 1, & \text{if } U \geq \theta \\ 0, & \text{otherwise.} \end{cases} \quad (6)$$

$$\text{if } \delta = 1, 0 \leftarrow U. \quad (7)$$

Unlike the non-spiking model, the synaptic conductance for spiking synapses G_{syn} is a dynamical variable. It is reset to a maximum value $G_{max,spike}$ whenever its corresponding pre-synaptic neuron spikes, and otherwise decays to 0 with a time constant τ_{syn} .

$$\tau_{syn} \frac{dG_{syn}}{dt} = -G_{syn} \quad (8)$$

$$\text{if } \delta = 1, G_{max,spike} \leftarrow G_{syn} \quad (9)$$

In order to perform more dynamic computation with spiking neurons, a mechanism for synaptic propagation delay can also be incorporated. If the synapse from neuron i to neuron j has a delay of d timesteps, the delayed spike can be defined as

$$\delta_{delay}^{j,i}[t] = \delta^{j,i}[d - \Delta t] \quad (10)$$

3 Software Design and Workflow

SNS-Toolbox allows for the design and implementation of synthetic nervous systems, and operates in three phases: design, compilation, and simulation.

3.1 Design Phase

When designing a network, users add neurons and populations of neurons to an overall *Network* object, with defined synaptic connections between the neurons. These connections can take on any topology desired, including feedback loops, and can be either individual synapses or patterns of synapses (an example of patterned synapses is given in Section 4.3). For applying external stimulus to a network, one-dimensional vectors can be added as input sources. To observe neural states during simulation, output monitors can be added. These monitors can be voltage-based or spike-based, for which the output is the direct voltage or spiking state of the source neuron.

This *Network* object by itself is not capable of being simulated, it merely acts as a defined storage container which describes the network parameters and structure. In the compilation phase, the object is referenced as a building plan to construct a network which can be simulated. Any network can also be used as an element within another network. In this way, a large network can be designed using large collections of predefined subnetworks. A collection of subnetworks that perform simple arithmetic and dynamic functions is available within SNS-Toolbox. For a complete explanation of these networks please refer to [21].

3.2 Compilation

Once a network is designed, it needs to be converted from a dictionary of parameters into an executable network. Given a *Network* object, SNS-Toolbox is able to build a new object which simulates a given network in one of four software backends: NumPy [12], PyTorch [18], a PyTorch-based sparse matrix library (torch.sparse), and a NumPy-based iterative evaluator which evaluates each synapse individually.

In order to improve simulation performance, the multiplication constant for the membrane voltage, firing threshold, and synapse dynamics are pre-computed. Instead of handling the timestep Δt and a time constant τ at each step, a time factor T is used instead where $T = \frac{\Delta t}{\tau}$.

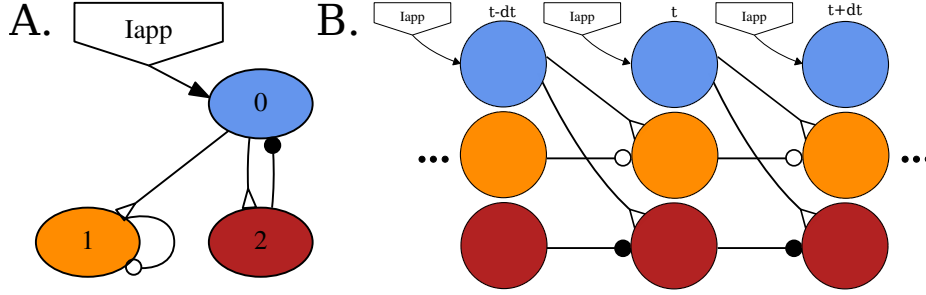


Fig. 1. Simulation method for a small example network using SNS-Toolbox. **A.** Overall network diagram generated within the toolbox. **B.** Diagram of the general computational flow when simulating the network.

3.3 Simulation

Commonly used high-performance frameworks focus on neural networks structured around massively connected individual layers [18], but SNS-Toolbox focuses on smaller networks connected with multiple levels of feedback loops [2, 14] found in animal nervous systems. Since these networks are more cyclic in nature, all networks simulated by SNS-Toolbox are unfolded through time, similar to the method pioneered by backpropagation through time [23]. For a visual representation of this unfolding, please see Figure 1. At every timestep, each neuron can receive synaptic input from any neuron at the previous timestep, including an autapse from itself. These connections retain the same properties from step to step, and these properties are stored in $N \times N$ matrices for a network of N neurons.

At a given timestep t , a user applies an external input vector $\vec{I}n[t]$. This vector could come from any source (e.g., static data, real-time sensors), but must be formatted in the dominant datatype of the software backend (either Numpy array or Torch tensor). Within the forward pass of simulation, first the memory states are updated as $\vec{U}_{last}[t] \leftarrow \vec{U}[t - \Delta t]$ and $\vec{\theta}_{last}[t] \leftarrow \vec{\theta}[t - \Delta t]$. Next the external input is applied to the correct neurons,

$$\vec{I}_{app}[t] \leftarrow \mathbf{C}_{in} \cdot \vec{I}n[t], \quad (11)$$

where $\mathbf{C}_{in}$ is an $L \times N$ (L is the number of input elements, N is the number of neurons in the network) binary masking matrix which routes each input element to its correct target neuron.

Adapting Equations 3 and 8, the non-spiking synaptic conductance matrix $\mathbf{G}_{non}$ and spiking counterpart $\mathbf{G}_{spike}$ are computed as

$$\mathbf{G}_{non}[t] \leftarrow \max \left(0, \min \left(\mathbf{G}_{max,non} \cdot \frac{\vec{U}_{last}[t]}{R}, \mathbf{G}_{max,non} \right) \right), \quad (12)$$

$$\mathbf{G}_{spike}[t] \leftarrow \mathbf{G}_{spike}[t - 1] \cdot (1 - \mathbf{T}_{syn}), \quad (13)$$

These conductances are summed to form the total synaptic conductance matrix $\mathbf{G}_{\text{syn}}[t]$. The synaptic current vector follows Equation 2, adapted to matrix form using row-wise sums

$$\vec{I}_{\text{syn}}[t] \leftarrow \sum_j \mathbf{G}_{\text{syn}}^{i,j}[t] \cdot \mathbf{E}^{i,j} - \vec{U}_{\text{last}}[t] \odot \sum_j \mathbf{G}_{\text{syn}}^{i,j}, \quad (14)$$

where $\odot$ denotes the element-wise Hadamard product. The new neural state $\vec{U}[t]$ and firing threshold $\vec{\theta}[t]$ are computed as

$$\vec{U}[t] \leftarrow \vec{U}_{\text{last}}[t] + \vec{T}_{\text{mem}} \odot \left(-\vec{G}_{\text{mem}} \odot \vec{U}_{\text{last}}[t] + \vec{I}_b + \vec{I}_{\text{syn}} + \vec{I}_{\text{app}} \right), \quad (15)$$

$$\vec{\theta}[t] \leftarrow \vec{\theta}_{\text{last}}[t] + \vec{T}_{\theta}[t] \odot \left(-\vec{\theta}_{\text{last}}[t] + \vec{\theta}_0 + \vec{m} \odot \vec{U}_{\text{last}}[t] \right). \quad (16)$$

Based on Equation 16, the spiking states are also updated,

$$\vec{\delta}[t] \leftarrow \text{sign} \left(\min \left(0, \vec{\theta} - \vec{U} \right) \right). \quad (17)$$

For ease of implementation, SNS-Toolbox internally represents spikes as impulses with a magnitude of -1.

Synaptic propagation delay is accomplished using a buffer matrix δ_{buffer} with N columns and D rows, where D is the longest delay time in timesteps within the network. At each timestep, the rows of δ_{buffer} are shifted down by 1, and the first row is replaced with the spike state vector of the current timestep $\vec{\delta}[t]$. A matrix of delayed spikes δ_{delay} is then generated by rearranging δ_{buffer} based on the amount of delay of each spiking synapse in the network. δ_{delay} is used to implement the spiking synapse reset dynamics in 9,

$$\mathbf{G}_{\text{spike}}[t] \leftarrow \max \left(\mathbf{G}_{\text{spike}}[t], -\delta_{\text{delay}}[t] \odot \mathbf{G}_{\text{max,spike}} \right). \quad (18)$$

Using the spike states, the neural membrane voltage of each neuron that spiked is reset to zero.

$$\vec{U}[t] \leftarrow \vec{U}[t] \odot \left(\vec{\delta}[t] + 1 \right) \quad (19)$$

The vector output is then computed,

$$\vec{Out}[t] \leftarrow \mathbf{C}_{\text{out,voltage}} \cdot \vec{U}[t] + \mathbf{C}_{\text{out,spike}} \cdot \vec{\delta}[t]. \quad (20)$$

This output is a general vector, which can subsequently be formatted or distributed to external systems based on the intention of the user.

Variants Each software backend implements the behavior of equations 11-20, but also has two variants available for improved performance. When no synaptic propagation delay is needed for any spiking synapses, one backend variant can be used where δ_{buffer} and δ_{delay} are never computed. The spiking state vector $\vec{\delta}[t]$ is used instead of δ_{delay} . Similarly, when there are no spiking neurons in the network a variant can be used where no spiking parameters or variables are computed and equations 13, 16-19 are not implemented. The performance difference of these variants is explored in Section 4.2.

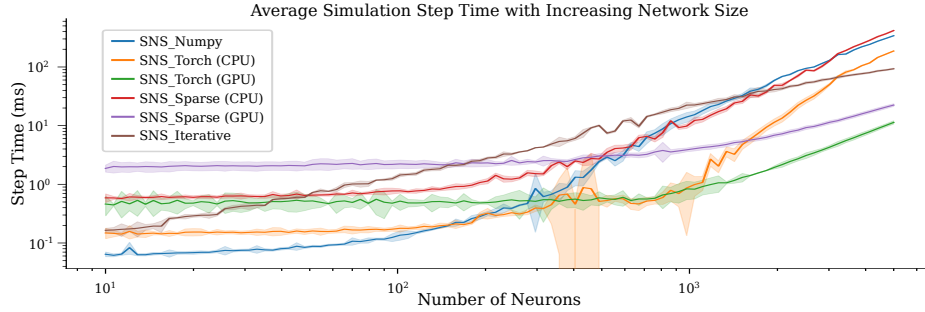


Fig. 2. Comparison of average wall-clock time to simulate a network for one simulation time-step. Solid lines represent the mean elapsed time per step over 100 steps, shaded areas show one standard deviation from the mean.

4 Results

4.1 Backend Simulation Performance

100 networks were run in a logarithmic spacing between 10 and 5000 spiking neurons, connected in the following structure: number of inputs = 8% of N , outputs = 12% of N , number of synapses = N . This structure is derived from a previous large-scale synthetic nervous system [14]. The mechanism for propagation delay was enabled. Each network was run for 100 steps, and the time to compute the forward pass was saved for each step. Results are shown in Figure 2. SNS_Numpy was the fastest until 158 neurons, then the CPU version of SNS_Torch until 358 neurons. Both versions of SNS_Torch are tied until 978 neurons, after which point the GPU version is the fastest.

Real-time Performance If SNS-Toolbox were to be interfaced with a real-world robotic system, a 1:1 correlation between the simulated and real step time (or better) would be desired. For non-spiking neurons with a time constant τ_{mem} of 5 ms, the coarsest timestep producing accurate dynamics would also be 5 ms. Looking at Figure 2, the largest network that could be theoretically simulated in real-time under these conditions would be 3,240 neurons, using the full GPU SNS_Torch backend. If the non-spiking variant were used, the maximum network size would be larger. For spiking networks, depending on the firing rate and synaptic dynamics the coarsest timestep will be smaller. Assuming a simulation timestep of 1 ms, the largest network which could be simulated would be 1,190 neurons. For step-times below 10 ms a relatively large variance can be observed, potentially due to interference from other processes in the operating system. Further benchmarking will be done on dedicated hardware systems with less overhead than a consumer desktop system.

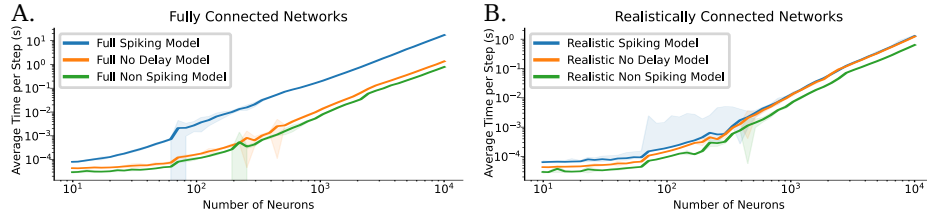


Fig. 3. Average wall-clock time to simulate a network for one simulation time-step, using three different variations of the SNS_Numpy backend. Network size was varied from 10 neurons to 10,000 neurons. Solid lines represent the mean elapsed time per step over 100 steps, shaded areas show one standard deviation from the mean. **A.** Networks are fully connected. **B.** The number of synaptic connections is equal to the number of neurons.

4.2 Backend Variant Performance

Isolating one backend family, we can examine the performance difference between different variants of the same backend. For this experiment, we chose to analyze SNS_Numpy. The number of neurons in a network was varied from 10 to 10,000, and the average timestep was measured for a backend implementing the full neural model, one without synaptic propagation delay, and one with no spiking whatsoever. The data is presented in Figure 3. For fully connected networks, after 20 neurons the full model is consistently one order of magnitude slower than the other models. When the network connectivity is more constrained, the performance difference between the backends reduces. Non-spiking is always the fastest, but for realistically connected networks the speed difference between the full model with and without synaptic propagation delay becomes relatively small.

4.3 Example Network Design

Using SNS-Toolbox, we implemented a network that models an anatomical circuit and performs a useful function, but would be complicated and tedious to manually route all of the synaptic connections. In the *Drosophila melanogaster* nervous system, the optic lobe contains circuits for processing motion in the visual field [3]. The first two layers of this visual system are the retina and the lamina, which perform two distinct visual processing operations. Retinal neurons R1-R6 encode incoming photons as changing neural activity [4], and the lamina primarily consists of two pathways: the L2 neurons have an antagonistic center-surround receptive field, and L1 neurons have a traditional center-surround receptive field [10]. Based on this structure, modeling the retina and L1 cells results in a circuit that performs high-pass filtering of an input image.

Previous work has created a synthetic nervous system model of the *Drosophila* optic lobe motion circuitry [19] using the Animatlab software [5]. This model was constrained to one-dimensional images, in part due to the difficulty of implementing the model. In this previous approach neurons and synapses had to

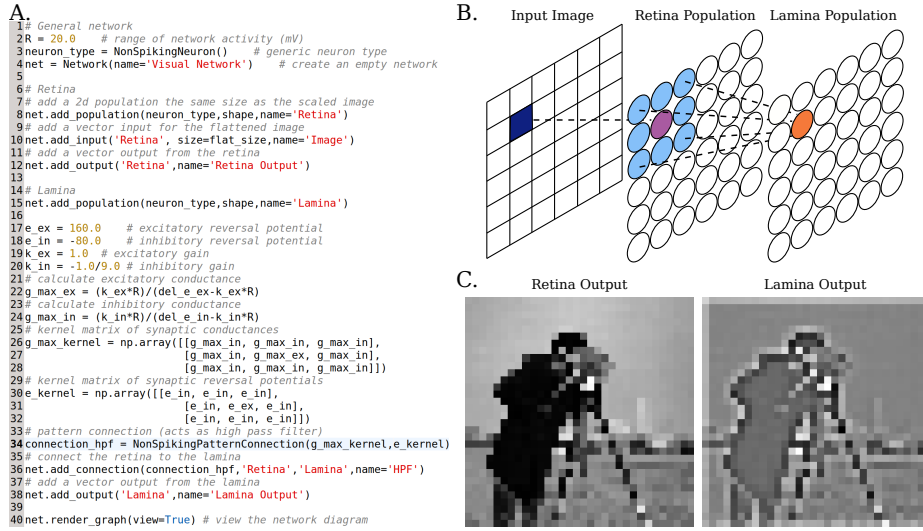


Fig. 4. Using SNS-Toolbox to design a two-layer visual processing system. **A.** Python code to generate the desired network. Image preprocessing and output plotting are omitted. **B.** Network visual representation. An input image is converted to stimulus current for a population of neurons, representing the insect retina. From the retina, a 3x3 kernel of inhibitory (light blue) and excitatory (purple) synapses is applied to create a high-pass filtering effect in the next layer, representing the L1 insect lamina neurons. **C.** Output of retina and lamina neurons, respectively. Voltages are mapped to grayscale intensities.

be placed and routed by hand, which is time-intensive and tedious to produce a simplified model with 510 neurons and 1574 synapses for one-dimensional images [19]. To demonstrate the effectiveness of SNS-Toolbox, we created a model of the retina and L1 cells of the optic lobe capable of processing two-dimensional images, consisting of 2048 neurons and 8476 synapses and generated in only 18 lines of Python code (see Figure 4A).

We assumed that input images are grayscale. Since a single *Drosophila* eye consists of around 800 ommatidia arranged in 32-34 columns [16], we also designed for input images which are 32x32 pixels. After creating two 32x32 populations of neurons and attaching an input source, the last step was to define the connection between the retina and the lamina. Our lamina model only consists of L1 neurons, so each neuron has center-on surround-off receptive field. Since each L1 cell receives input from the directly adjacent ommatidia, we can implement these receptive fields as a 3x3 connection kernel:

$$\mathbf{K} = \begin{bmatrix} k_{in} & k_{in} & k_{in} \\ k_{in} & k_{ex} & k_{in} \\ k_{in} & k_{in} & k_{in} \end{bmatrix},$$

where k_{in} and k_{ex} are the desired gains for inhibitory and excitatory synapses, respectively. For desired behavior, these gains were chosen as $k_{in} = -\frac{1}{9}$ and $k_{ex} = 1$. These synaptic gains are transformed into synaptic conductances and relative reversal potentials using the method described in [21].

The results of simulating the network are shown in Figure 4. The output of the lamina layer correctly implements a rudimentary high-pass filter or edge-detector, as expected.

5 Discussion & Future Work

In this work, we introduced SNS-Toolbox, a software tool for designing and simulating recurrent networks of biologically-inspired spiking and non-spiking neurons with conductance-based synapses. For networks with a few hundred neurons, the CPU-based backends are the fastest simulation approach; for networks with more neurons, GPU-based backends are fastest.

There are existing software packages [13, 15] for simulating conductance-based neurons and synapses, but are primarily designed for offline simulation. Methods to interface these simulators with robotic hardware are impractical [20]. Other software packages are efficient simulators, but not designed to efficiently simulate conductance-based neurons [15, 18]. Others still simulate networks of neurons, but these packages are inherently tied to physics engines and overhead such that it is impossible to simulate large networks in real-time or faster [5, 9]. Future work will provide quantitative speed comparisons between SNS-Toolbox and other simulation systems.

SNS-Toolbox implements two specific neural models commonly used in the SNS literature, and aims to simulate them as fast as possible on consumer computer hardware. Networks can contain both of these models, but cannot be expanded by the user to include new models since SNS-Toolbox is a specific accelerator for these models. Larger models could be approximated by treating these neurons as compartments within a larger model, an approach commonly employed with neuromorphic hardware [1], but this has not been explored at this time. These neural models are also lacking any voltage-gated ion-specific channels, which can be used to create pattern-generators [11]. Future work will implement an optional voltage-gated ion channel for these oscillatory networks.

Section 4.3 details the design and simulation of an example network which performs vision processing. While this network demonstrates some of the features of SNS-Toolbox, the example is not the best to showcase the strengths of the toolbox over other software. Future work will use SNS-Toolbox to implement motor systems, an area for which SNS-Toolbox is better suited.

In this work, we focus on using SNS-Toolbox to simulate networks of neurons in isolation. At its heart, SNS-Toolbox takes an input vector at each timestep and generates an output vector. These inputs and outputs can be arbitrary data (as they are in this work), but could include data streams from a robot, camera, or any continuous source. Future work will implement interfaces for interaction with physics simulators, as well as an interface for external robotic hardware. In its

current state the whole codebase is developed in Python for ease of maintenance and development. In the future, the addition of a C++-based backend would allow for additional improvements in performance.

References

1. Lava software framework (2021)
2. Beer, R.D., Gallagher, J.C.: Evolving dynamical neural networks for adaptive behavior. *Adaptive Behavior* **1**, 91–122 (1992)
3. Borst, A.: *Drosophila's view on insect vision* (1 2009)
4. Clark, D.A., Demb, J.B.: Parallel computations in insect and mammalian visual motion processing (10 2016)
5. Cofer, D., Cymbalyuk, G., Reid, J., Zhu, Y., Heitler, W.J., Edwards, D.H.: AnimateLab: A 3d graphics environment for neuromechanical simulations. *Journal of Neuroscience Methods* **187**, 280–288 (3 2010)
6. Cohen, G.: Gooaaall!!!: Why we built a neuromorphic robot to play foosball. *IEEE Spectrum* **59**, 44–50 (3 2022)
7. Eliasmith, C., Anderson, C.H.: *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. MIT press (2003)
8. Eshraghian, J.K., Ward, M., Neftci, E., Wang, X., Lenz, G., Dwivedi, G., Benamoun, M., Jeong, D.S., Lu, W.D.: Training spiking neural networks using lessons from deep learning (9 2021)
9. Falotico, E., Vannucci, L., Ambrosano, A., Albanese, U., Ulbrich, S., Tieck, J.C.V., Hinkel, G., Kaiser, J., Peric, I., Denninger, O., Cauli, N., Kirtay, M., Roennau, A., Klinker, G., Arnim, A.V., Guyot, L., Peppicelli, D., Mactinaz-Cañada, P., Ros, E., Maier, P., Weber, S., Huber, M., Plecher, D., Röhrbein, F., Deser, S., Roitberg, A., Smagt, P.V.D., Dillman, R., Levi, P., Laschi, C., Knoll, A.C., Gewaltig, M.O.: Connecting artificial brains to robots in a comprehensive simulation framework: The neurorobotics platform. *Frontiers in Neurobotics* **11** (1 2017)
10. Freifeld, L., Clark, D.A., Schnitzer, M.J., Horowitz, M.A., Clandinin, T.R.: Gabaergic lateral interactions tune the early stages of visual processing in *Drosophila*. *Neuron* **78**, 1075–1089 (6 2013)
11. Goldsmith, C.A., Szczecinski, N.S., Quinn, R.D.: Neurodynamic modeling of the fruit fly *Drosophila melanogaster*. *Bioinspiration & Biomimetics* **15**, 065003 (9 2020)
12. Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M.H., Brett, M., Haldane, A., del Río, J.F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., Oliphant, T.E.: Array programming with numpy (9 2020)
13. Hines, M.L., Carnevale, N.T.: Neuron: A tool for neuroscientists (2001), <http://www.neu>
14. Hunt, A., Szczecinski, N., Quinn, R.: Development and training of a neural controller for hind leg walking in a dog robot. *Frontiers in Neurobotics* **11** (4 2017)
15. Kulkarni, S.R., Parsa, M., Mitchell, J.P., Schuman, C.D.: Benchmarking the performance of neuromorphic and spiking neural network simulators. *Neurocomputing* **447**, 145–160 (8 2021)
16. Kumar, J.P.: Building an ommatidium one cell at a time (1 2012)

17. Mozafari, M., Ganjtabesh, M., Nowzari-Dalini, A., Masquelier, T.: Spyketorch: Efficient simulation of convolutional spiking neural networks with at most one spike per neuron. *Frontiers in Neuroscience* **13** (2019)
18. Paszke, A., Gross, S., Massa, F., Lerer, A., Google, J.B., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Xamla, A.K., Yang, E., Devito, Z., Nabla, M.R., Tejani, A., Chilamkurthy, S., Ai, Q., Steiner, B., Facebook, L.F., Facebook, J.B., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. Curran Associates, Inc. (2019)
19. Sedlackova, A., Szczecinski, N.S., Quinn, R.D.: A synthetic nervous system model of the insect optomotor response (2020)
20. Strohmer, B., Manoonpong, P., Larsen, L.B.: Flexible spiking cpgs for online manipulation during hexapod walking. *Frontiers in Neurorobotics* **14** (6 2020)
21. Szczecinski, N.S., Hunt, A.J., Quinn, R.D.: A functional subnetwork approach to designing synthetic nervous systems that control legged robot locomotion. *Frontiers in Neurorobotics* **11** (8 2017)
22. Szczecinski, N.S., Quinn, R.D., Hunt, A.J.: Extending the functional subnetwork approach to a generalized linear integrate-and-fire neuron model. *Frontiers in Neurorobotics* **14** (11 2020)
23. Werbos, P.J.: Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE* **78** (1990)