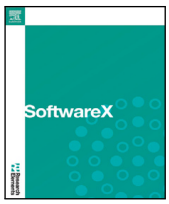
SoftwareX 27 (2024) 101759

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx



Original software publication

WiN-GUI: A graphical tool for neuron-based encoding

Simon F. Müller-Cleve [a,*], Fernando M. Quintana [b], Vittorio Fra [c], Pedro L. Galindo [b], Fernando Perez-Peña [b], Gianvito Urgese [c], Chiara Bartolozzi [a]

[a] EDPR, Istituto Italiano di Tecnologia, Genoa, Italy
[b] School of Engineering, University of Cádiz, Cádiz, Spain
[c] Politecnico di Torino, Turin, Italy

ARTICLE INFO

Keywords:
Robotics
Neuromorphic
Spike encoding
Development tool
Graphical user interface

ABSTRACT

Neuromorphic computing relies on event-based, energy-efficient communication, inherently implying the need for conversion between real-valued (sensory) data and binary, sparse spiking representation. This is usually accomplished by using the real-valued data as current input to a spiking neuron model and tuning the neuron's parameters to match a desired – often biologically inspired – behavior. To support the investigation of neuron models and parameter combinations to identify suitable configurations for neuron-based encoding of sample-based data into spike trains we developed the WiN-GUI. Thanks to the generalized LIF model implemented by default, next to the LIF and Izhikevich neuron models, many spiking behaviors can be investigated out of the box offering the possibility of tuning biologically plausible responses to the input data. The GUI is provided open source and with documentation and is easy to extend with further neuron models and personalize with data analysis functions.

Code metadata

Current code version	v1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-23-00764
Permanent link to Reproducible Capsule	https://github.com/event-driven-robotics/win-GUI/releases/tag/v1.0
Legal Code License	GPL-3.0
Code versioning system used	git
Software code languages, tools, and services used	Matplotlib, Pandas, Numpy, PyDub, PyQt6, Python, PyTorch, scikit-learn, SciPy
Compilation requirements, operating environments & dependencies	Requires PyQT6 and PyTorch with Python. Tested on Linux and Windows
If available Link to developer documentation/manual	
Support email for questions	simon.mullercleve@iit.it

1. Motivation and significance

The realm of event-based – or spike-based – encoding and computation has garnered substantial attention in recent times, evident from the proliferation of publications and advancements in hardware development for sensing [1–4] and processing [5–8]. This surge in interest has mainly focused on event-driven vision, with prominent industry players such as Sony [9] and Intel [10] venturing into this domain.

A noteworthy advantage, alongside the attainment of low-latency performance, is the possibility of obtaining reduced energy consumption and memory footprint. Despite the expanding foundation of both software and hardware resources, the event-based sensing capabilities of the research community remain mostly focused in the domain of vision as a consequence of the availability of native event-driven sensors. This constraint arises from multiple factors. One of these is the demanding and time-intensive nature of hardware development, which requires a deep understanding of both design and fabrication processes on the one hand and lossless spike-encoding on the other hand. While in the vision domain, it is generally accepted that the event-driven encoding is implemented as change detection, in other sensory modalities like touch, more complex neuron models with closer behavior to biology are used [11–13].

* Corresponding author.
E-mail address: simon.mullercleve@iit.it (Simon F. Müller-Cleve).

https://doi.org/10.1016/j.softx.2024.101759
Received 16 January 2024; Received in revised form 17 April 2024; Accepted 3 May 2024
Available online 22 May 2024

Fig. 1. The WiN-GUI: on the left-hand side are the plots used to visualize the input data and analyze the neuron dynamics (a, b), on the right-hand side are the data (c, d), channel (e), parameter (f), and audio (g) settings.

The availability of a tool capable of visualizing and measuring the effects of parameters and diverse neuron models with varying encoding schemes is important to support the exploration of encoding strategies and their effects on event-based computation. It could potentially catalyze the discovery of novel applications. Addressing these challenges head-on, we introduce the Watch inside Neurons - GUI (WiN-GUI). Specifically intended for exploration and development of neuron-based encoding strategies, it stands apart from static analyses of algorithm-based encoding techniques [14–17]. The Graphical User Interface (GUI) offers a flexible environment to interactively investigate different neuronal behaviors and to customize the spike-encoding process.

2. Software description

2.1. Software architecture

The GUI is developed in Python3 and employs PyQT6 for visualization and PyTorch for neuron modeling. It is accessible via Github.[1] Comprising six primary components – data visualization, data selection, preprocessing, channel selection, neuron model and parameter selection, and audio output – it offers a comprehensive toolset for designing and analyzing spiking neurons. The visual surface allows fast and easy access to all necessary variables. The fast and responsive interface enables easy parameter sweeping while monitoring the impact on the screen or headphones.

2.2. Software functionalities

2.2.1. Data visualization

This segment presents three critical elements: the sample-based input signal, the internal state variable traces of neurons, and the spike raster plot depicting the network's spiking activity. The display is structured with the sample-based data showcased at the top-left, the neuron membrane potential at the top-right, and the remaining state variables in a $2 \times N$ grid below, found in Fig. 1a. Furthermore, a compact spike raster plot is positioned at the bottom for rapid spike pattern assessment, highlighted in Fig. 1b. For enhanced clarity, the color per channel is synchronized across all plots, simplifying mapping between different visualization elements. The visualization process relies on the Matplotlib library, enabling straightforward adaptation and customization, including the use of various color palettes.

2.2.2. Data format and selection

The GUI currently supports data in the form of a Pandas *DataFrame* containing a list of individual Python dictionaries, adhering to a specific naming pattern. If the dataset involves multiple classes or repetitions of classes, each trial should contain entries for *class* and *repetition*, if applicable. If both *class* and *repetition* are applicable, the data selection panel shown in Fig. 1c becomes accessible, otherwise only the given subset. The top section displays the loaded file's name, which can be interactively changed using a file browser. The bottom right area houses a combo box for class selection (if indicated by the *class* key), while a dial on the left enables repetition selection (if indicated by the *repetition* key).

2.2.3. Preprocessing

The preprocessing section is located in Fig. 1d. Preprocessing options encompass four checkboxes and two sliders. The top left checkbox enables normalization, and the one right to it enables signal filtering using multidimensional image processing from the SciPy library. The checkbox below at the left sets the starting point of each sensor to zero, and the right splits each sensor channel into two sub-channels, one containing only previously positive values and the other containing the rectified values of previously negative readings. Missing values are filled with zeros. This transformation proves advantageous when the data contains negative sensor readings, which usually lead to a decrease in the neuron's membrane potential, possibly preventing any spiking activity. The sample-based signal is normalized per sensor across all trials. The two sliders below are for re-sampling and re-scaling of the input signal, changing the temporal dynamics of the stimulus. To maintain the GUI's responsiveness, the visualization remains independent of the re-sampling factor. That is, while updating the neuron model

[1] https://github.com/event-driven-robotics/win-GUI

simulation, the visualized data is down-sampled to match the original sampling rate. Notably, the simulation time step for neuronal dynamics matches the input data sampling rate and can be adjusted using the re-sample slider.

2.2.4. Channel selection

The channel selection feature enables users to hide selected channels for a clean view of the remaining, simplifying inspection. The sensor layout can be customized to accommodate individual preferences and is shown in Fig. 1e.

2.2.5. Neuron model and encoding parameters

The GUI extends support to a wide range of neuron models written in PyTorch, contingent upon adhering to a specific format definition. The currently supported models include the neuron proposed by Mihalaş-Niebur in [18], the neuron proposed by Izhikevich [19], the Leaky Integrate-and-Fire (LIF) and Recurrent Leaky Integrate-and-Fire (RLIF) neuron model from [20]. It is possible to integrate any model, regardless of the number of state variables, as long as the *neuronStateVariable* starts with the neuron's membrane potential voltage V and ends with the tensor containing the neuron's spikes spk. All further state variables must be listed between these two. Additionally, to enable the exchange of manipulable parameters between the neuron simulation and the GUI, a dictionary named *parameters_dict* must be provided. This dictionary enumerates all parameters intended for manipulation. It is crucial to assign these parameters before running the GUI, ensuring successful synchronization of slider values with neuron parameter updates. The dictionary includes specifications such as the lowest and highest slider values, step size, and start value. The neuron model selection and parameter slider are located in Fig. 1f. Each parameter specified in the mentioned dictionary will be listed and can be manipulated according to the range and step size specified. For parameters not listed, the default value will be used.

2.2.6. Audio output

Positioned at Fig. 1g, this section governs the audio output of the spike raster plot. The audio output allows auditory inspection of the spiking pattern to complement visual examination. Using the PyDub library in Python, the GUI generates ticks using the sawtooth function, thereby maintaining separation between multiple adjacent ticks and ensuring a distinct auditory experience.

3. Illustrative examples

One compelling example involves recreating the 20 distinct neuron behaviors of the generalized LIF model as outlined in [18]. The GUI is equipped with additional examples, presenting input current traces alongside tactile Braille data. In Table 1, four representative parameter sets and their corresponding inputs I_e/C are listed. Fig. 2 visually represents these sets, serving as exemplary use cases. These sets share three parameter configurations and three input current configurations. Notably, *M: Tonic bursting* and *O: Rebound burst* share parameters for different inputs, while *M: Tonic bursting* and *P: Mixed mode* have different parameters for the same input. Each case unveils distinct firing patterns, enabling the analysis of the neuron's internal dynamics.

Example 1. The same parameters for *M: Tonic bursting* and *O: Rebound burst* induce diverse neuronal responses. A constant excitatory (positive) input I_e/C results in tonic bursting, while an inhibitory (negative) step input triggers a brief burst. This underscores that the neuron's behavior is influenced not only by its inherent parameters but also by the nature of the input it receives.

Analyzing the membrane voltage trace V and firing threshold Thr provides insights into this behavior. In *O: Rebound burst*, the membrane potential exponentially returns to its resting state along with the threshold variable. At $t = 0.65$, when $Thr = -0.07$ and $V = -0.07$, the membrane potential surpasses the threshold, initiating bursting. This continues until the high inhibitory current i_2 prevents further spikes, evident in the membrane potential drop at $t = 0.7$. Subsequently, the threshold variable remains above the membrane potential, preventing any additional response.

Example 2. Comparing *M: Tonic bursting* and *P: Mixed mode* under the same constant input of 2 I_e/C reveals distinctive firing patterns. In *M: Tonic bursting*, frequent burst events occur with a decreasing number of spikes in each burst, while in *P: Mixed mode*, the same input leads to frequent spiking with an increase in Interspike intervals (ISIs). Both firing patterns reach a steady state towards the end.

The GUI allows us to manipulate parameters between these sets and observe the resulting changes. Starting with the parameter set of *M: Tonic bursting* and decreasing A_1/C to match the values of *P: Mixed mode*, the bursting behavior gradually fades out. At $A_1/C = 6.5$, only the first three spikes come in a burst, followed by single spikes with an increasing ISI. A further reduction to null, or below, results in single spikes with a constantly increasing ISI. Conversely, starting again from the parameter set of *M: Tonic bursting* and increasing A_2/C to match *P: Mixed mode*, the bursts contain more spikes and last longer, appearing more regularly as the constant input persists. For this input and parameter set, A_1/C marks the transition from bursting to frequency adaptation, with values below null being the lower limit for bursting.

Example 3. The sole distinction, aside from the parameter a, lies in the input when examining *L: Hyperpolarizing bursting* and *M: Tonic bursting*, where $a = 30$ and excitatory input applies to the former, and $a = 5$ with inhibitory input for the latter. Despite both inducing burst events, the underlying mechanics differ significantly.

In *L: Hyperpolarizing bursting*, with inhibitory input, the threshold variable Thr drops faster than the membrane potential. Eventually, it surpasses the membrane potential, initiating the first bursting event. The initial elicited spike resets the threshold to a higher value above the membrane potential, leading to subsequent spikes and causing the burst. During this period, the threshold variable recovers and exceeds the membrane potential threshold. Subsequently, the inhibitory current lowers the threshold variable, completing the cycle. All subsequent bursts occur with the same frequency and time difference, following the same dynamics.

Conversely, in *M: Tonic bursting*, with excitatory input, the threshold variable monotonically increases. Similarly, the membrane potential, upon surpassing the threshold, leads to spikes and an increasing negative current i_2, reducing the membrane potential until it falls below the threshold. From this point, the input current is integrated again, the threshold variable continues to rise, and the next bursting event is triggered with the described dynamics. However, it occurs at a higher membrane voltage due to the elevated threshold variable. The number of spikes per burst decreases, eventually settling into a repetitive pattern once the threshold variable saturates.

The three described examples show 1. how beneficial it is to be able to have all state variables in a compact view to trace down the interplay between them, 2. how easy parameter sweeps can be realized to find conditional boundaries, and 3. how easy different conditions (input traces) and parameter sets can be explored.

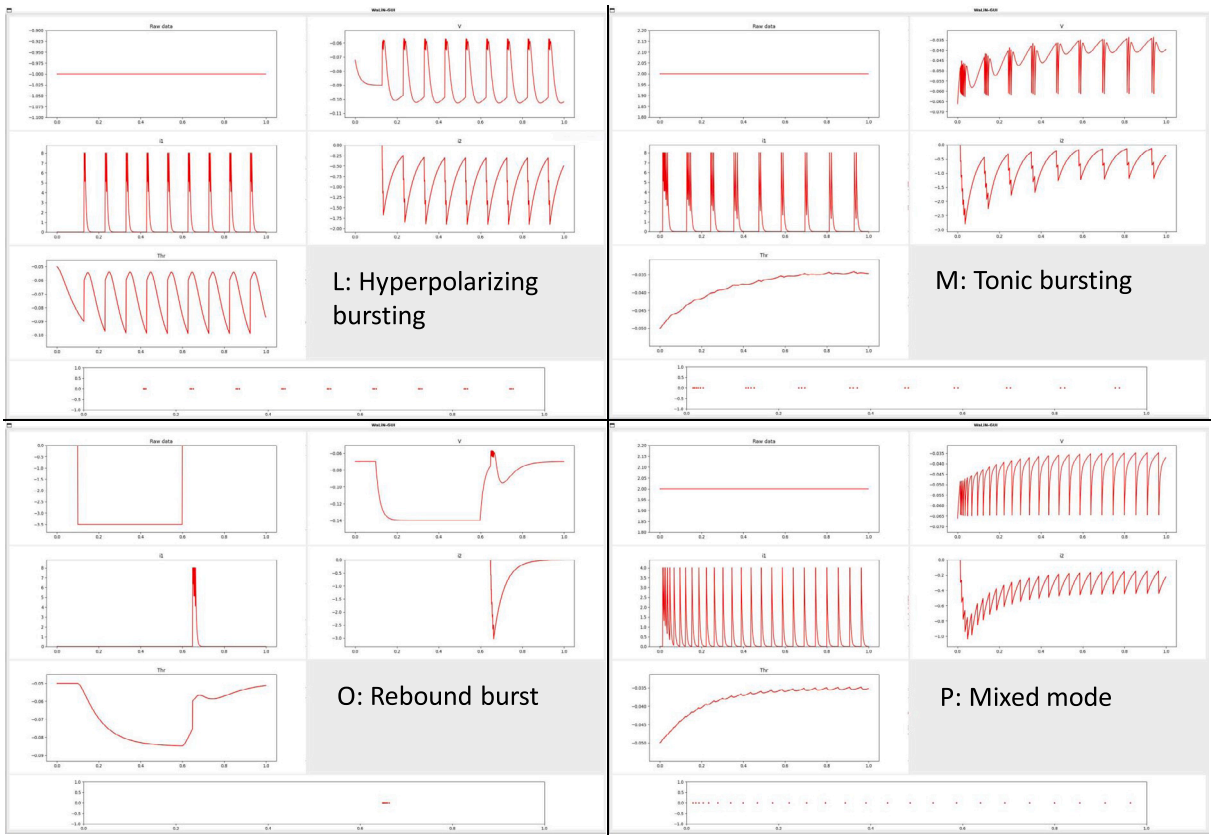


Fig. 2. Four different neuron behaviors presented in *A Generalized Linear Integrate-and-Fire Neural Model Produces Diverse Spiking Behavior* by Ş. Mihalaş and E. Niebur. (Top-left) Hyperpolarizing bursting, (Top-right) Tonic bursting, (Bottom-left) Rebound bursting and (Bottom-right) Mixed mode, showcasing the impact of input and parameter interplay.

Table 1
Parameters for the four example neuron behaviors. Other free parameters are set to $b = 10\ \text{s}^{-1}$, $G/C = 50\ \text{s}^{-1}$, $k_1 = 200\ \text{s}^{-1}$, $k_2 = 20\ \text{s}^{-1}$, $R_1 = 0$, and $R_2 = 1$.

Parameter	a [s^{-1}]	A_1/C [V/s]	A_2/C [V/s]	I_e/C [V/s]
L: Hyperpolarizing bursting	30	10	−0.6	−1
M: Tonic bursting	5	10	−0.6	2
O: Rebound burst	5	10	−0.6	0, −3.5, 0
P: Mixed mode	5	5	−0.3	2

4. Impact

While various GUIs for modeling neurons and neural networks, offering the ability to manipulate parameters, already exist, none of these effectively facilitates the utilization of recorded experimental data in a tailored manner for the task at hand. Most existing GUIs serve educational purposes within biology classrooms, while others focus on investigating interactions between groups of neurons. A notable mention is the potent *NEURON* [21] simulator, which allows the definition of diverse neuron models, dynamic adjustment of their parameters, modification of network topology, and a plethora of other functions. This simulation framework strives to replicate *in vivo* cell growth and connectivity development, resulting in high realism but also high complexity. Custom data input is not anticipated, although it supports fundamental encoding analysis functions. Conversely, the *neuronModelGUI* [22] and *NESIM-RT* [23] serve as tools to scrutinize the interactions among multiple neurons of various types. It enables straightforward, intuitive placement of neurons and connections with adjustable parameters. The integration of recorded datasets into neurons' inputs is currently uncertain. Additionally, the reliance on Java and C++ coding could hinder the adoption of Python-based neuron models. This presents an issue as many machine learning practitioners prefer Python and utilize PyTorch for training Spiking Neural Networks (SNNs). Unfortunately, none of the existing alternatives support neuron simulations with the PyTorch framework, despite some being Python-based.

In contrast, our proposed GUI is designed on purpose to empower developers interested in neuron-based encoding. Its primary goal is to aid in identifying the most suitable neuron model and its associated parameters for converting sequences of sample-based real values into temporally sparse discrete event-based data. Achieving this objective is facilitated through an interactive interface that provides comprehensive information at a glance. Furthermore, the GUI incorporates auditory inspection, allowing users to assess the impact of neuron parameter adjustments on population firing patterns. To illustrate its efficacy, we used example data from the tactile Braille data, acquired using a robotic touch-sensitive fingertip, as presented in [24]. In addition to the well-known Izhikevich and LIF neuron models, we included the generalized LIF model described by Ş. Mihalaş and E. Niebur to showcase diverse neuronal responses and the ramifications of distinct neuron parameters.

5. Known limitations and future work

A limitation of the GUI is the complexity involved in integrating new models. Currently, users must provide the new neuron model in one file, manipulatable parameters in another, and include the name in the Python file running the GUI. Although the process is documented, it poses a risk of errors due to the multiple files and required actions. While it is feasible to include customized models by modifying the software code, a future goal is to simplify this process by creating a model template, possibly through an API. This would enable easier implementation of additional neuron models without altering the GUI's internal code. For example, such models could be imported in PTB format via WiN-GUI's graphical interface. Additionally, considering

intermediate representations of neuron models through computational primitives, as proposed by NIR [25], could allow inspecting neurons from various frameworks and hardware platforms directly within the GUI.

The GUI serves as a tool to monitor neuron responses, analyzing how neuron models react to specific inputs. Unlike SNNs simulators such as NESIM-RT or NEURON, the GUI focuses solely on single-neuron analysis without complex features like neuron connections. However, it offers a user-friendly approach to examining individual neuron responses based on input data and model parameters. If there is interest in expanding its capabilities to include additional layers and Machine Learning (ML) functionalities, leveraging the underlying PyTorch library makes it feasible.

As future improvements to the software, we plan to enhance dataset loading and preprocessing by providing users with a standard dataset format, reducing complexity and avoiding the need for internal code modifications. This will enhance the user-friendliness of the WiN-GUI interface. Furthermore, we aim to incorporate additional preprocessing techniques, such as trial-wise or time-step-wise normalization. Finally, we plan to introduce a new audio output option, allowing users to choose between playing selected channels or all channels.

6. Conclusions

In conclusion, the WiN-GUI provides a powerful and versatile interface for researchers and developers working with neuron-based encoding and SNNs. Its array of features spans data selection, visualization, preprocessing, channel configuration, neuron model integration, and tuning, with audio output. This creates a comprehensive toolkit for designing, exploring, and understanding neural encoding by accounting for specific needs depending on the target application. We believe this is a useful addition to the tools available to researchers in the neuromorphic computation and sensing domain and it will grow with the contribution of the community.

CRediT authorship contribution statement

Simon F. Müller-Cleve: Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Fernando M. Quintana:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Vittorio Fra:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Pedro L. Galindo:** Writing – review & editing, Supervision. **Fernando Perez-Peña:** Writing – review & editing, Supervision. **Gianvito Urgese:** Writing – review & editing, Supervision. **Chiara Bartolozzi:** Writing – review & editing, Project administration, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Link to data and code in manuscript. All open access.

Acknowledgments

The authors extend their gratitude to the participants and organizers of the 2022 Telluride neuromorphic workshop for fostering invaluable discussions and providing a wellspring of inspiration. A special acknowledgment is reserved for the Neuromorphic Tactile Exploration (NTE) group, whose contributions greatly enriched the workshop experience. S.F.M.-C. is supported by the European Union's Horizon 2020 MSCA Programme under Grant Agreement No. 813713 *NeuTouch*. F.M.Q. is supported by the FPU grant (FPU18/04321) from the Spanish Ministry of Universities and the Spanish national research project *NEMOVISION: Sistemas Neuromórficos para Visión Artificial* (PID2019-109465RB-I00). V.F. acknowledges funding by the European Union - NextGenerationEU Project *3A-ITALY* (PE0000004, CUP E13C22001900001). P.L.G. is supported by the Spanish national research project *NEMOVISION: Sistemas Neuromórficos para Visión Artificial* (PID2019-109465RB-I00). F.P. is supported by the Spanish grant (with support from the European Regional Development Fund) MINDROB (PID2019-105556GB-C33). G.U. acknowledges a contribution from the Italian National Recovery and Resilience Plan (NRRP), M4C2, funded by the European Union – NextGenerationEU (Project IR0000011, CUP B51E22000150006, *EBRAINS-Italy*).

References

[1] Lichtsteiner P, Posch C, Delbruck T. A 128×128 120 dB 15μs latency asynchronous temporal contrast vision sensor. IEEE J Solid-State Circuits 2008;43(2):566–76. http://dx.doi.org/10.1109/JSSC.2007.914337.

[2] Brandli C, Berner R, Yang M, Liu S-C, Delbruck T. A 240×180 130 db 3μs latency global shutter spatiotemporal vision sensor. IEEE J Solid-State Circuits 2014;49(10):2333–41. http://dx.doi.org/10.1109/JSSC.2014.2342715.

[3] Son B, Suh Y, Kim S, Jung H, Kim J-S, Shin C, et al. 4.1 A 640×480 dynamic vision sensor with a 9μm pixel and 300Meps address-event representation. 2017, p. 66–7. http://dx.doi.org/10.1109/ISSCC.2017.7870263.

[4] Posch C, Matolin D, Wohlgenannt R. A QVGA 143 dB dynamic range frame-free PWM image sensor with lossless pixel-level video compression and time-domain CDS. IEEE J Solid-State Circuits 2010;46(1):259–75. http://dx.doi.org/10.1109/JSSC.2010.2085952.

[5] Davies M, Srinivasa N, Lin T-H, Chinya G, Cao Y, Choday SH, et al. Loihi: A neuromorphic manycore processor with on-chip learning. IEEE Micro 2018;38(1):82–99. http://dx.doi.org/10.1109/MM.2018.112130359.

[6] Moradi S, Qiao N, Stefanini F, Indiveri G. A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs). IEEE Trans Biomed Circuits Syst 2017;12(1):106–22. http://dx.doi.org/10.1109/TBCAS.2017.2759700.

[7] Mayr C, Hoeppner S, Furber S. Spinnaker 2: A 10 million core processor system for brain simulation and machine learning. 2019, http://dx.doi.org/10.48550/arXiv.1911.02385, arXiv preprint arXiv:1911.02385.

[8] Orchard G, Frady EP, Rubin DBD, Sanborn S, Shrestha SB, Sommer FT, et al. Efficient neuromorphic signal processing with loihi 2. In: 2021 IEEE workshop on signal processing systems. siPS, IEEE; 2021, p. 254–9. http://dx.doi.org/10.1109/SiPS52927.2021.00053.

[9] Pelé A-F. Event-driven vision hits production lines. EE Times 2019. URL www.eetimes.com/event-driven-vision-hits-production-lines.

[10] Lin C-K, Wild A, Chinya GN, Cao Y, Davies M, Lavery DM, et al. Programming spiking neural networks on Intel's Loihi. Computer 2018;51(3):52–61. http://dx.doi.org/10.1109/MC.2018.157113521.

[11] Saal HP, Delhaye BP, Rayhaun BC, Bensmaia SJ. Simulating tactile signals from the whole hand with millisecond precision. Proc Natl Acad Sci USA 2017;144(28). http://dx.doi.org/10.1073/pnas.1704856114.

[12] Soni M, Dahiya R. Soft eSkin: distributed touch sensing with harmonized energy and computing. Phil Trans R Soc A 2019;378(2164). http://dx.doi.org/10.1098/rsta.2019.0156.

[13] Bergner F, Mittendorfer P, Dean-Leon E, Cheng G. Event-based signaling for reducing required data rates and processing power in a large-scale artificial robotic skin. In: 2015 IEEE/RSJ international conference on intelligent robots and systems. IROS, 2015, p. 2124–9. http://dx.doi.org/10.1109/IROS.2015.7353660.

[14] Petro B, Kasabov N, Kiss RM. Selection and optimization of temporal spike encoding methods for spiking neural networks. IEEE Trans Neural Netw Learn Syst 2019;31(2):358–70. http://dx.doi.org/10.1109/TNNLS.2019.2906158.

[15] Auge D, Hille J, Mueller E, Knoll A. A survey of encoding techniques for signal processing in spiking neural networks. Neural Process Lett 2021;53(6):4693–710. http://dx.doi.org/10.1007/s11063-021-10562-2.

[16] Schuman C, Rizzo C, McDonald-Carmack J, Skuda N, Plank J. Evaluating encoding and decoding approaches for spiking neuromorphic systems. In: Proceedings of the international conference on neuromorphic systems 2022. 2022, p. 1–9. http://dx.doi.org/10.1145/3546790.3546792.

[17] Forno E, Fra V, Pignari R, Macii E, Urgese G. Spike encoding techniques for IoT time-varying signals benchmarked on a neuromorphic classification task. Front Neurosci 2022;16:999029. http://dx.doi.org/10.3389/fnins.2022.999029.

[18] Mihalaş Ş, Niebur E. A generalized linear integrate-and-fire neural model produces diverse spiking behaviors. Neural Comput 2009;21(3):704–18. http://dx.doi.org/10.1162/neco.2008.12-07-680.

[19] Izhikevich E. Simple model of spiking neurons. IEEE Trans Neural Netw 2003;14(6):1569–72. http://dx.doi.org/10.1109/TNN.2003.820440.

[20] Cramer B, Stradmann Y, Schemmel J, Zeke F. The Heidelberg spiking data sets for the systematic evaluation of spiking neural networks. IEEE Trans Neural Netw Learn Syst 2022;33. http://dx.doi.org/10.1109/TNNLS.2020.3044364.

[21] Hines ML, Carnevale NT. NEURON: a tool for neuroscientists. Neurosci: Rev J Bringing Neurobiol Neurol Psychiatry 7(3):123–35. http://dx.doi.org/10.1177/107385840100700207.

[22] Christie I. NeuronModelGUI. URL https://github.com/iankchristie/NeuronModelGUI.

[23] Rosa-Gallardo DJ, de la Torre JC, Quintana FM, Dominguez-Morales JP, Perez-Peña F. NESIM-RT: A real-time distributed spiking neural network simulator. SoftwareX 2023;22:101349. http://dx.doi.org/10.1016/j.softx.2023.101349.

[24] Müller-Cleve SF, Fra V, Khacef L, Pequeño-Zurro A, Klepatsch D, Forno E, et al. Braille letter reading: A benchmark for spatio-temporal pattern recognition on neuromorphic hardware. Front Neurosci 2022;16. http://dx.doi.org/10.3389/fnins.2022.951164.

[25] Pedersen JE, Abreu S, Jobst M, Lenz G, Fra V, Bauer FC, et al. Neuromorphic intermediate representation: A unified instruction set for interoperable brain-inspired computing. 2023, http://dx.doi.org/10.48550/arXiv.2311.14641, arXiv:2311.14641. URL https://neuroir.org/.