The Journal of Open Source Software

BlueCelluLab: Biologically Detailed Neural Network Experimentation API

Anıl Tuncel[1], **Werner Van Geit**[1,3], **Mike Gevaert**[1], **Benjamin Torben-Nielsen**[1,4], **Darshan Mandge**[1], **İlkan Kılıç**[1], **Aurélien Jaquier**[1], **Eilif Muller**[1,5,6], **Lida Kanari**[1], **and Henry Markram**[1,2]

1 Blue Brain Project, École Polytechnique Fédérale de Lausanne (EPFL), Campus Biotech, 1202 Geneva, Switzerland **2** Laboratory of Neural Microcircuitry (LNMC), Brain Mind Institute, School of Life Sciences, École Polytechnique Fédérale de Lausanne (EPFL), 1015 Lausanne, Switzerland **3** Present address -> Foundation for Research on Information Technologies in Society (IT'IS), Zurich 8004, Switzerland **4** Present address -> Roche Information Solutions, F. Hoffmann-La Roche AG, Basel, Switzerland **5** Present address -> Université de Montréal, CHU Ste-Justine Research Center (Architectures of Biological Learning Lab), Montréal, Canada **6** Present address -> Mila Quebec AI Institute, Montréal, Canada

DOI: 10.21105/joss.07026

Software

- Review
- Repository
- Archive

Editor: Beatriz Costa Gomes

Reviewers:

- @finsberg
- @ryEllison

Submitted: 19 June 2024
Published: 17 August 2024

Summary

The NEURON simulator, established in 1984 and continuously developed since, stands as the preeminent tool for neuron simulation within computational neuroscience. Its widespread adoption and compatibility with computational clusters and supercomputers underscore its pivotal role in large-scale neuronal research. However, its integration with the Python programming language has introduced complexities, particularly concerning memory management and object lifecycle. To conceal these challenges from the user and seamlessly interface with community standards for neural network representation data formats such as SONATA, we introduce BlueCelluLab. The high-level Python API simplifies the execution of neural simulations, ranging from single neurons to intricate networks, by managing complexities related to memory management and object lifecycle, thus providing a streamlined experience for users. Today, BlueCelluLab is powering various Python packages, command line interfaces, web applications, and data analysis workflows.

Statement of Need

The NEURON simulator has been a cornerstone in computational neuroscience for decades and is now the most frequently used neuron simulator in the field (Tikidji-Hamburyan et al., 2017). The NEURON simulator is renowned for its high efficiency in running large-scale simulations and maintains strong compatibility with computational clusters and modern supercomputer architectures, making it well-suited for utilisation across parallel computing environments (Awile et al., 2022; Kumbhar et al., 2019).

Initially, the NEURON simulator was programmed using the hoc (higher order calculator) interpreted programming language (Kernighan & Pike, 1984). Years later, a Python interface was developed to facilitate communication with the NEURON simulator, making it accessible to Python users (Hines et al., 2009). The NEURON package, though resembling a typical Python package and importable like many others, exhibits significant operational differences. One major difference is that the scope and lifetime of NEURON objects do not align with those of typical Python objects. If not properly managed, these discrepancies can lead to memory safety vulnerabilities that compromise system security, consume excessive resources, and potentially corrupt scientific results. This places an undue burden on scientists to manage technical details beyond their experimental focus.

Moreover there is a need for an interface that can load and simulate neural networks from the SONATA data format (Dai et al., 2020) jointly developed by the Allen Institute and the Blue Brain Project. BlueCelluLab's API supports loading and simulating SONATA networks with the NEURON simulator, offering a highly configurable environment. Users can load the entire network or selectively load subsets tailored to specific scientific inquiries. For example, if a study focuses on the behavior of a particular neuronal electrical type, users can select and simulate only those specific neurons from the SONATA network.

BlueCelluLab complements the Blue Brain Project's large-scale simulator neurodamus (Pereira et al., 2024), which runs NEURON simulations in parallel using MPI. This integration allows for efficient replay simulations in BlueCelluLab, where researchers can adjust neuron properties and rerun parts of simulations generated by neurodamus. This method saves resources and enables the observation of how changes in a neuron or group of neurons influence their behavior within the entire network.

System Overview

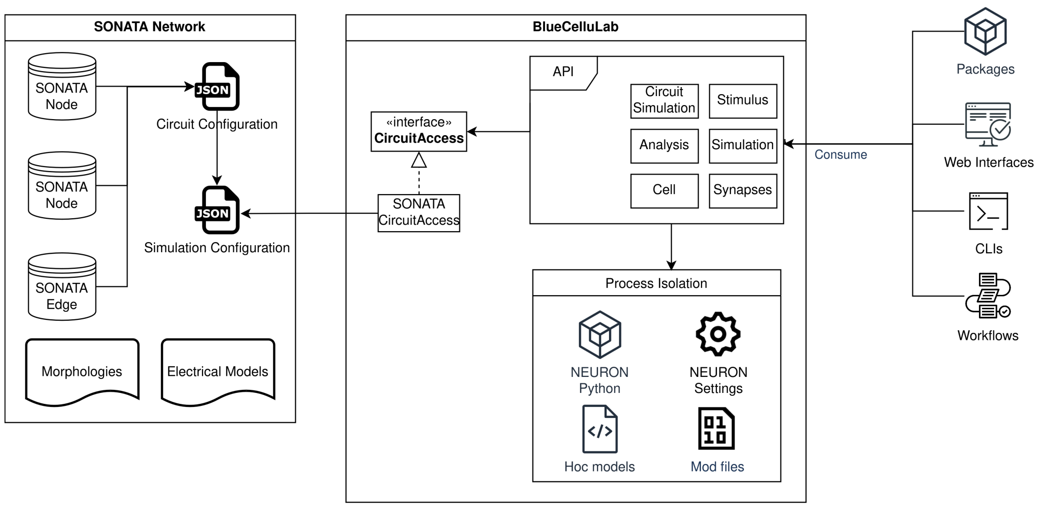

Figure 1: Architecture of BlueCelluLab

The system overview, as illustrated in Figure 1, shows that BlueCelluLab is built on top of the NEURON Python package, providing a high-level API tailored for the common use cases of single neuron or neural network simulations. The figure depicts the key components and interactions within BlueCelluLab, highlighting the modular design that facilitates integration and extension.

BlueCelluLab employs an IsolatedProcess mechanism to run simulations in separate processes when required. For example, prior to simulating neurons, users may need to conduct preliminary simulations to calculate specific properties of the neurons that will be utilised during the main simulation.

The network representation in BlueCelluLab is not confined to the SONATA format. With the use of dependency inversion principle (Martin, 2003), any network format that implements the CircuitAccess interface is seamlessly supported without requiring any integration effort at the API level. Currently, there are two distinct implementations available: one for SONATA and another for an internal format used by the Blue Brain Project.

External systems, such as other Python packages, web backends, CLIs, or software workflows, interact with the high-level API without needing to know its detailed implementation, thus embodying the principle of information hiding (Parnas, 1972).

BlueCelluLab covers common error scenarios to prevent crashes and improve user experience, enhancing system robustness significantly. Additionally, it is designed to fail gracefully in cases of misusage, further safeguarding against disruptions and maintaining a smooth operational flow.

The code snippet below demonstrates the initialisation and execution of a neural simulation using BlueCelluLab.

```
from bluecellulab.cell import create_ball_stick
from bluecellulab import Simulation

cell = create_ball_stick()
sim = Simulation()
sim.add_cell(cell)
stimulus = cell.add_step(start_time=5.0, stop_time=20.0, level=0.5)

sim.run(25)
time, voltage = cell.get_time(), cell.get_soma_voltage()
```

A neuron model is created using the create_ball_stick function, representing a basic type of neuron model. A simulation environment is initialised with Simulation(), and the created cell is added to this environment using sim.add_cell(cell). Next, a stimulus is attached to the neuron, activating it between 5.0 ms and 20.0 ms with a stimulation level of 0.5 nA (nanoamperes). The simulation is run for a total of 25 ms. Finally, time and soma voltage data are retrieved for further analysis and visualisation.

BlueCelluLab interfaces with various type systems, including those from Python, NEURON, network data formats, and dependencies such as numeric computing or data frame packages. To ensure type safety, it employs PEP-484 type annotations (Van Rossum et al., 2014) and a static type checker (Lehtosalo et al., 2017) for early error detection during static analysis. For dynamic input lacking type information at static time, BlueCelluLab uses the Pydantic Data Validation package (Colvin et al., 2024) to identify errors at runtime, thereby enhancing system robustness.

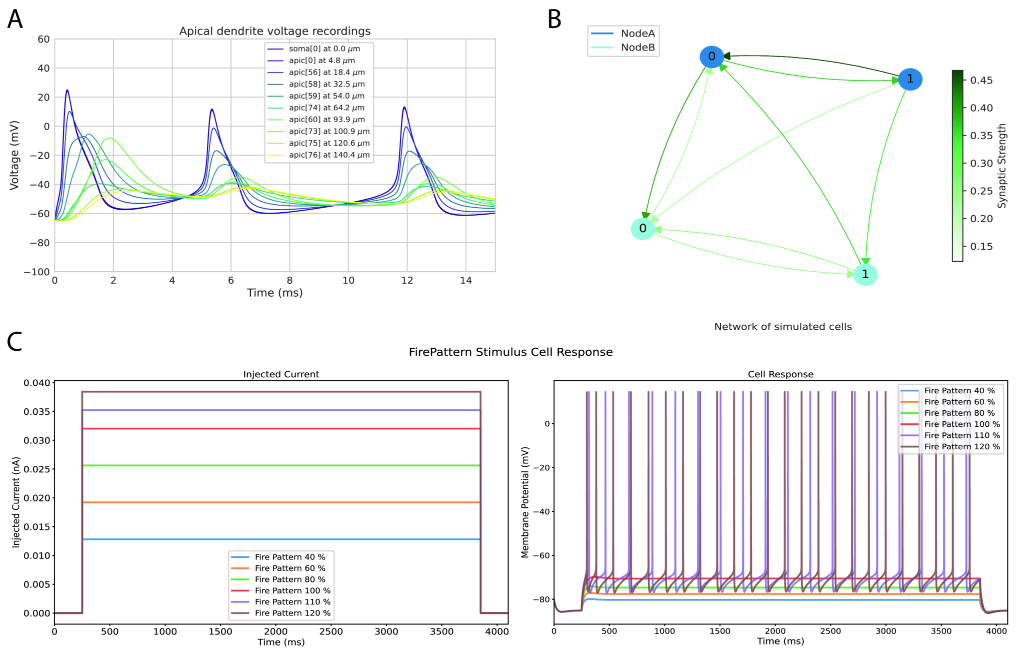

Figure 2: Applications

Figure 2: Illustrative examples of applications using bluecellulab: (A) Voltage recordings from the soma and various apical dendrite sections of a simulated pyramidal neuron, showcasing distinct potential changes across different cellular regions. (B) A graphical representation of a neuronal network, where edge colours indicate connection strengths and node types correspond to different neural subnetworks, such as the hippocampus and thalamus. (C) Responses of a neuron to different step current injection. The left figure displays the injected current and the right figure demonstrate variations in voltage behavior and spiking frequency relative to the amplitude of the applied currents that are expressed as percentages of rheobase current of the simulated neuron.

We believe that while BlueCelluLab is focused on computational neuroscience, the ideas that shaped its design decisions will also prove beneficial in research software across various multidisciplinary fields.

Acknowledgements

This study was supported by funding to the Blue Brain Project, a research center of the École polytechnique fédérale de Lausanne (EPFL), from the Swiss government's ETH Board of the Swiss Federal Institutes of Technology.

References

Awile, O., Kumbhar, P., Cornu, N., Dura-Bernal, S., King, J. G., Lupton, O., Magkanaris, I., McDougal, R. A., Newton, A. J. H., Pereira, F., Săvulescu, A., Carnevale, N. T., Lytton, W. W., Hines, M. L., & Schürmann, F. (2022). Modernizing the NEURON simulator for sustainability, portability, and performance. *Frontiers in Neuroinformatics*, *16*. https://doi.org/10.3389/fninf.2022.884046

Colvin, S., Jolibois, E., Ramezani, H., Badaracco, A. G., Dorsey, T., Montague, D., Matveenko, S., Trylesinski, M., Runkle, S., Hewitt, D., & Hall, A. (2024). *Pydantic* (Version v2.6.4).

https://docs.pydantic.dev/latest/

Dai, K., Hernando, J., Billeh, Y. N., Gratiy, S. L., Planas, J., Davison, A. P., Dura-Bernal, S., Gleeson, P., Devresse, A., Dichter, B. K., Gevaert, M., King, J. G., Geit, W. A. H. van, Povolotsky, A. V., Muller, E., Courcol, J. D., & Arkhipov, A. (2020). The SONATA data format for efficient description of large-scale network models. *PLoS Computational Biology*, *16*. https://doi.org/10.1371/journal.pcbi.1007696

Hines, M. L., Davison, A. P., & Muller, E. (2009). NEURON and python. *Frontiers in Neuroinformatics*, *3*. https://doi.org/10.3389/neuro.11.001.2009

Kernighan, B. W., & Pike, R. (1984). *The UNIX programming environment* (Vol. 270). Prentice-Hall Englewood Cliffs, NJ. https://doi.org/10.1002/spe.4380090102

Kumbhar, P., Hines, M., Fouriaux, J., Ovcharenko, A., King, J., Delalondre, F., & Schürmann, F. (2019). CoreNEURON : An optimized compute engine for the NEURON simulator. *Frontiers in Neuroinformatics*, *13*. https://doi.org/10.3389/fninf.2019.00063

Lehtosalo, J., Rossum, G. v, Levkivskyi, I., Sullivan, M. J., Fisher, D., Price, G., Lee, M., Seyfer, N., Barton, R., Ilinskiy, S., & others. (2017). Mypy-optional static typing for python. *URL: Https://Mypy-Lang.org*.

Martin, R. C. (2003). *Agile software development: Principles, patterns, and practices*. Prentice Hall PTR. https://doi.org/10.1002/pfi.21408

Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, *15*(12), 1053–1058. https://doi.org/10.1007/978-3-642-59412-0_26

Pereira, F., Ji, W., Magkanaris, I., Alonso, J. B., Kumbhar, P., Săvulescu, A., Heeren, E., Sergio, Bellotta, A., delalondre, fabien, Wolf, M., Geit, W. V., Lupton, O., Grosheintz, L., Temerev, A., Cornu, N., Awile, O., fschuerm, JCGoran, … bbp-hpcteam. (2024). *BlueBrain/neurodamus: 3.1.0* (Version 3.1.0). Zenodo. https://doi.org/10.5281/zenodo.10809263

Tikidji-Hamburyan, R. A., Narayana, V., Bozkus, Z., & El-Ghazawi, T. A. (2017). Software for brain network simulations: A comparative study. *Frontiers in Neuroinformatics*, *11*. https://doi.org/10.3389/fninf.2017.00046

Van Rossum, G., Lehtosalo, J., & Langa, L. (2014). Pep 484–type hints. *Index of Python Enhancement Proposals*.