# The NeuroML ecosystem for standardized multi-scale modeling in neuroscience

Ankur Sinha, Padraig Gleeson, Bóris Marin, Salvador Dura-Bernal, Sotirios Panagiotou, Sharon Crook, Matteo Cantarelli, Robert C Cannon, Andrew P. Davison, Harsha Gurnani, et al.

Neuroscience

# The NeuroML ecosystem for standardized multi-scale modeling in neuroscience

**Reviewed Preprint**

Published from the original preprint after peer review and assessment by eLife.

About eLife's process

**Reviewed preprint version 1**
May 3, 2024 (this version)

**Sent for peer review**
January 31, 2024

**Posted to preprint server**
December 11, 2023

**Ankur Sinha, Padraig Gleeson, Bóris Marin, Salvador Dura-Bernal, Sotirios Panagiotou, Sharon Crook, Matteo Cantarelli, Robert C. Cannon, Andrew P. Davison, Harsha Gurnani, R. Angus Silver**

Department of Neuroscience, Physiology and Pharmacology, University College London, United Kingdom • Universidade Federal do ABC, São Bernardo do Campo, Brazil • State University of New York, Brooklyn, USA • Center for Biomedical Imaging and Neuromodulation, Nathan S. Kline Institute for Psychiatric Research, Orangeburg, NY, USA • Erasmus Medical Center, Rotterdam, Netherlands • Arizona State University, USA • MetaCell Ltd. • Opus2 International Ltd, UK • CNRS, Gif-sur-Yvette, France • University of Washington, Seattle, USA



## Abstract

Data-driven models of neurons and circuits are important for understanding how the properties of membrane conductances, synapses, dendrites and the anatomical connectivity between neurons generate the complex dynamical behaviors of brain circuits in health and disease. However, the inherent complexity of these biological processes make the construction and reuse of biologically-detailed models challenging. A wide range of tools have been developed to aid their construction and simulation, but differences in design and internal representation act as technical barriers to those who wish to use data-driven models in their research workflows. NeuroML, a model description language for computational neuroscience, was developed to address this fragmentation in modeling tools. Since its inception, NeuroML has evolved into a mature community standard that encompasses a wide range of model types and approaches in computational neuroscience. It has enabled the development of a large ecosystem of interoperable open source software tools for the creation, visualization, validation and simulation of data-driven models. Here, we describe how the NeuroML ecosystem can be incorporated into research workflows to simplify the construction, testing and analysis of standardized models of neural systems, and supports the FAIR (Findability, Accessibility, Interoperability, and Reusability) data principles, thus promoting open, transparent and reproducible science.

**eLife assessment**

This **important** work presents a consolidated overview of the NeuroML2 open community standard. It provides **convincing** evidence for its central role within a broader comprehensive software ecosystem for the development of neuronal models that are open, shareable, reproducible, and interoperable. A major strength of the presented work is the persistence of the development over more than two decades to establish, maintain, and adapt this standard to meet the evolving needs of the field. This work is of broad interest to the sub-cellular, cellular, computational, and systems neuroscience communities undertaking studies involving theory, modeling, and simulation.

# Introduction

Development of an in-depth, mechanistic understanding of brain function in health and disease requires different scientific approaches spanning multiple scales, from gene expression to behavior. While "wet" experimental approaches are essential for characterizing the properties of neural systems and testing hypotheses, theory and modeling are critical for exploring how these complex systems behave across a wider range of conditions, and for generating new experimentally testable, physically plausible hypotheses. Theory and modeling also provide a way to integrate a panoply of experimentally measured parameters, functional properties, and responses to perturbations into a physio-chemically coherent framework that reproduces the properties of the neural system of interest (***Einevoll et al., 2019***; ***Yao et al., 2022***; ***Poirazi and Papoutsi, 2020***; ***Gurnani and Silver, 2021***; ***Gleeson et al., 2018***; ***Cayco-Gajic et al., 2017***; ***Billings et al., 2014***; ***Vervaeke et al., 2010***; ***Kriener et al., 2022***; ***Billeh et al., 2020***; ***Markram et al., 2015***).

Computational models in neuroscience often focus on different levels of description. For example, a cellular physiologist may construct a complex multicompartmental model to explain the dynamical behavior of an individual neuron in terms of its morphology, biophysical properties, and ionic conductances (***Hay et italic., 2011***; ***De Schutter and Bower, 1994***; ***Migliore et al., 2005***). In contrast, to relate neural population activity to sensory processing and behavior, a systems neurophysiologist may build a circuit level model consisting of thousands of much simpler integrate-and-fire neurons (***Lapicque, 1907***; ***Potjans and Diesmann, 2014***; ***Brunel, 2000***). Domain specific tools have been developed to aid the construction and simulation of models at varying levels of biological detail and scales. An ecosystem of diverse tools is powerful and flexible, but it also creates a number of serious challenges for the research community (***Cannon et al., 2007***). Each tool typically has its own design, features, Application Programming Interface (API) and syntax, a custom set of utility libraries, and last but not least, a distinct machine-readable representation of the model's physiological components. This represents a complex landscape for users to navigate. Additionally, models developed in different simulators cannot be mixed and matched, nor easily compared, and the translation of a model from one tool-specific implementation to another can be non-trivial and error-prone. This fragmentation in modeling tools and approaches can act as a barrier to neuroscientists who wish to use models in their research, as well as impede how Findable, Accessible, Interoperable, and Reusable (FAIR) models are (***Wilkinson et al., 2016***).

To counter fragmentation and promote cooperation and interoperability within and across fields, standardization is required. The International Neuroinformatics Co-ordinating Facility (INCF) (***Abrams et al., 2022***) has highlighted the need for standards to "make research outputs

machine-readable and computable and are necessary for making research FAIR" (*INCF, 2023*). In biology, a number of community standards have been developed to describe experimental data (e.g. Brain Imaging Data Structure (BIDS) (*Gorgolewski et al., 2016*), Neurodata Without Borders (NWB) (*Teeters et al., 2015*)) and computational models (e.g. Systems Biology Markup Language (SBML) (*Hucka et al., 2003*), CellML (*Lloyd et al., 2004*), Scalable Open Network Architecture TemplAte (SONATA) (*Dai et al., 2020*), PyNN (*Davison et al., 2009*) and Neural Open Markup Language (NeuroML) (*Gleeson et al., 2010*)). These standards have enabled open and interoperable ecosystems of software applications, libraries and databases to emerge, facilitating the sharing of research outputs, an endeavor encouraged by a growing number of funding agencies and scientific journals.

The initial version of the NeuroML standard, version 1 (NeuroMLv1), was originally conceived as a model description format (*Goddard et al., 2001*), and implemented as a three layered, declarative, modular, simulator independent language (*Gleeson et al., 2010*). NeuroMLv1 could describe detailed neuronal morphologies and their biophysical properties as well as specific instantiations of networks. It enabled the archiving of models in a standardized format and addressed the issue of simulator fragmentation by acting as the common language for model exchange between established simulation environments—NEURON (*Hines and Carnevale, 1997*; *Awile et al., 2022*), GENESIS (*Bower and Beeman, 1997*), and MOOSE (*Ray and Bhalla, 2008*). While solving a number of long-standing problems in computational neuroscience, NeuroMLv1 had several key limitations. The most restrictive of these was that the dynamical behavior of model elements was not formally described in the standard itself, making it only partially machine readable. Information on the dynamics of elements (i.e. how the state variables should evolve in time) was only provided in the form of human-readable documentation, requiring the developers of each new simulator to re-implement the behavior of these elements in their native format. Additionally, introduction of new model components required updates to the standard and all supporting simulators, making extension of the language difficult. Finally, the use of Extensible Markup Language (XML) as the primary interface language limited usability—applications would generally have to add their own code to read/write XML files.

To address these limitations, NeuroML was redesigned from the ground up in version 2 (NeuroMLv2) using the Low Entropy Modeling Specification (LEMS) language (*Cannon et al., 2014*). LEMS was designed to define a wide range of physio-chemical systems, enabling the creation of fully machine-readable, formal definitions of the structure and dynamics of any model components. The core modeling elements in NeuroMLv2 (cells, ion channels, synapses) have their mathematical and structural definitions described in LEMS (e.g. the parameters required and how the state variables change with time). Thus, NeuroMLv2 retains all the features of NeuroMLv1—it remains modular, declarative, and continues to support multiple simulation engines—but unlike version 1, it is extensible, and all specifications are fully machine-readable. NeuroMLv2 also moved to Python as its main interface language and provides a comprehensive set of Python libraries to improve usability (*Vella et al., 2014*), with XML retained as a machine-readable serialization format (i.e. the form in which the model files are saved/shared).

Since its release in 2014, the NeuroMLv2 standard, the software ecosystem, and the community have all steadily grown. An open, community based governance structure was put in place—an elected Editorial Board, overseen by an independent Scientific Committee, maintains the standard and core tools. Although the core tools were initially focused on enabling the simulation of models on multiple platforms, they have been expanded to support all stages of the model life cycle (**Fig. 1**). Modelers can use these tools to easily create, inspect and visualize, validate, simulate, fit and optimize, share and disseminate NeuroMLv2 models and outputs (*Billings et al., 2014*; *Cayco-Gajic et al., 2017*; *Gurnani and Silver, 2021*; *Kriener et al., 2022*; *Gleeson et al., 2019*). To provide clear, concise, searchable information for both users and developers, the NeuroML documentation has been significantly expanded and re-deployed using the latest modern web technologies[1]. Increased community-wide collaborations have also extended the software

ecosystem well beyond the NeuroMLv2 tools developed by the core NeuroML team: additional simulators such as Brian (***Stimberg et al., 2019***), NetPyNE (***Dura-Bernal et al., 2019***), Arbor (***Abi Akar et al., 2019***) and EDEN (***Panagiotou et al., 2022***) all support NeuroMLv2. We have worked to ensure interoperability with other structured formats for model development in neuroscience such as PyNN (***Davison et al., 2009***) and SONATA (***Dai et al., 2020***). Platforms for collaboratively developing, visualizing, and sharing NeuroML models (Open Source Brain (OSB) (***Gleeson et al., 2019***)) as well as a searchable database of NeuroML model components (NeuroML Database (NeuroML-DB) (***Birgiolas et al., 2023***)) have been developed. These enhancements, driven by an ever expanding community, have helped NeuroMLv2 mature into a standard which has been officially endorsed by international organizations such as the INCF and COmputational Modeling in Biology NEtwork (COMBINE) (***Hucka et al., 2015***), and which is now sufficiently mature to be incorporated into a wide range of research workflows.

In this paper, we provide an overview of the current scope of version 2 of the NeuroML standard, describe the current software ecosystem and community, and outline the extensive resources to assist researchers to incorporate NeuroML into their modeling work. We demonstrate, with examples, that NeuroML supports users in all stages of the model development life cycle (**Fig. 1**) and promotes FAIR principles in computational neuroscience. We highlight the various core NeuroML tools and libraries, additional utilities, supported simulation engines, and the related projects that build upon NeuroML for automated model validation, advanced analysis, visualization, and sharing/re-use of models. Finally, we summarize the organizational aspects of NeuroML, its governance structure and community.

# Results

## NeuroML provides a ready to use set of curated model elements

A central aim of the NeuroML initiative is to enable and encourage the use of multi-scale biophysically detailed models of neurons and neuronal circuits in neuroscience research. The initiative takes a number of steps to achieve this aim. NeuroML provides users with a curated library of model elements that form the NeuroML standard[2] (**Fig. 2**). The standard is maintained by the NeuroML Editorial Board which has identified a core set of model types to support, to ensure that a significant proportion of commonly used neurobiological modeling entities can be described with the language. This includes (but is not limited to): active membrane conductances (using Hodgkin-Huxley style (***Hodgkin and Huxley, 1952***) or kinetic scheme based ionic conductances), multiple synapse and plasticity mechanisms, detailed multicompartmental neuron models with morphologies and biophysical properties, abstract point neuron models, and networks of such cells spatially arranged in populations, connected by targeted projections, receiving spiking and current based inputs.

The NeuroMLv2 standard consists of two levels that are designed to enable users to easily create their models without worrying about simulator-specific details. The first level defines a formal “schema” for the standard model elements, their attributes/parameters (e.g. an integrate and fire cell model and its necessary attributes: a threshold parameter, a reset parameter, etc.), and the relationships between them (e.g. a network contains populations; a multicompartmental cell morphology contains segments). This allows the validation of the completeness of the description of individual NeuroML model elements and models, *prior to simulation*. The second level defines the underlying dynamical behavior of the model elements (e.g. how the time varying membrane potential of a cell model is to be calculated). Most users do not need to interact with this level (which is enabled by LEMS), which, among other features, facilitates the automated translation of *simulator-independent* NeuroML models into *simulator-specific* code.

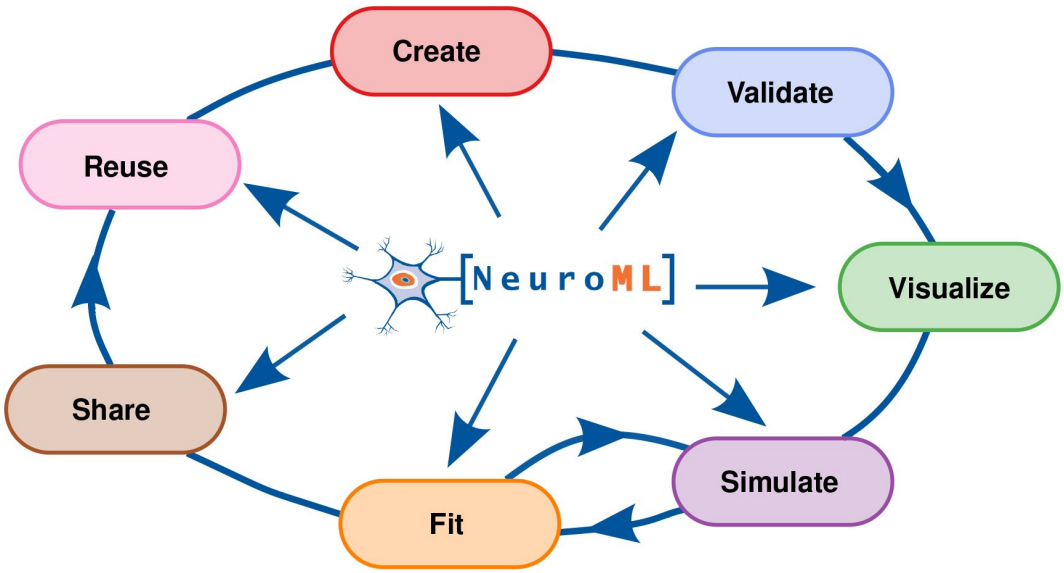
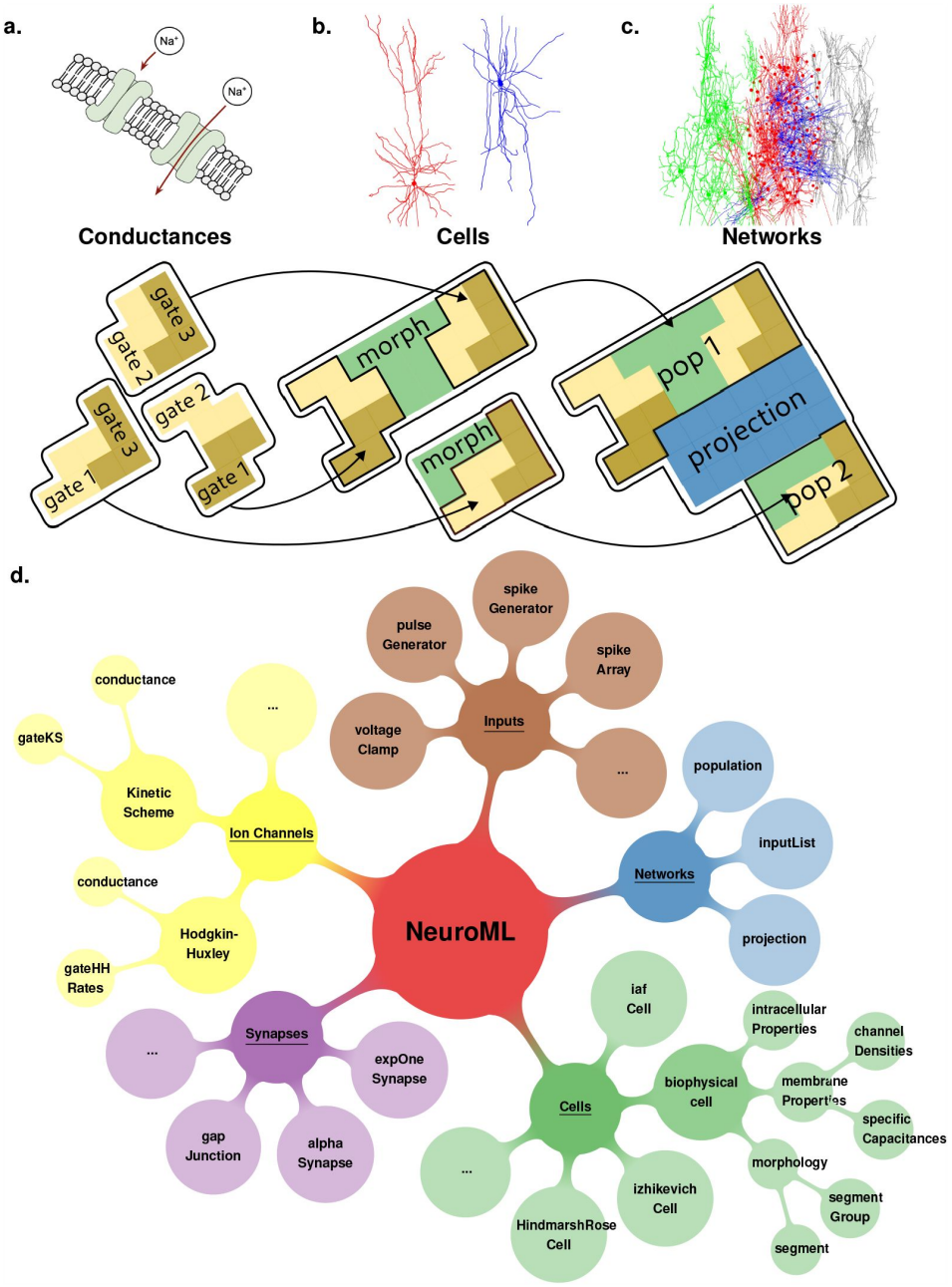



**Figure 1.**

The NeuroML software ecosystem supports all stages of the model development life cycle.




## Figure 2.

NeuroML is a modular, hierarchical format that supports multi-scale modeling. Elements in NeuroML are formally defined, independent, self-contained building blocks with hierarchical relationships between them. **(a)** Models of ensembles of **ionic conductances** can be defined as a composition of gates, each with specific voltage (and potentially [Ca2+]) dependence that controls the conductance. **(b)** Morphologically detailed **neuronal models** specify the 3D structure of the cells, along with passive electrical properties, and reference ion channels that confer membrane conductances. **(c) Network models** contain populations of these cells connected via synaptic projections. **(d)** A truncated illustration of the NeuroMLv2 core element hierarchy. The standard includes commonly used model elements/building blocks that have been pre-defined for users in a hierarchical representation: **Cells**: neuronal models ranging from simple spiking point neurons to biophysically detailed cells with multicompartmental morphologies and active membrane conductances; **Synapses and ionic conductance models**: commonly used chemical and electrical synapse models (gap junctions), and multiple representations for ionic conductances; **Inputs**: to drive cell and network activity, e.g. current or voltage clamp, spiking background inputs; **Networks**: of populations (containing any of the aforementioned cell types), and projections between them, form the core of a NeuroML model description. The full list of NeuroML elements can be found in **Appendix 1 Tables 4** and **5**.

Thus, modelers can use the standard NeuroML elements to conveniently build simulator-independent models, while also being able to examine and extend the underlying implementations of models. As a simulator independent language, NeuroML also promotes interoperability between different computational modeling tools, and as a result, the standard library is complemented by a large, well maintained ecosystem of software tools that supports all stages of the model life cycle—from creation, analysis, simulation and fitting, to sharing and reuse. Finally, as discussed in later sections, for advanced use cases where the existing NeuroML model building blocks are insufficient, NeuroML also includes a framework for creating and including new model elements.

## NeuroML is a modular, structured language for defining FAIR models

NeuroMLv2 is a modular, structured, hierarchical, simulator independent format. All NeuroML elements are formally defined, independent, self-contained elements with hierarchical relationships between them. An "ionic conductance" model element in NeuroML, for example, can contain zero, one or more "gates" and plug into a "cell" model element, which can then fit into a "population" of a "network" (**Fig. 2**). To support the range of electrical properties found in biological neurons, ionic conductances with distinct ionic selectivities and dynamics can be generated in NeuroML through the inclusion of different types of gates (e.g. activation, inactivation), their dependence on variables such as voltage and $[Ca^{2+}]$ and their reversal potential. Cell types with different functional and biophysical properties can then be generated by conferring combinations of ionic conductances on their membranes. The conductance density can be adjusted to generate the electrophysiological properties found *in vivo*. In practice, many examples of ionic conductances that underlie the electrical behavior of neurons are already available in NeuroMLv2 and can simply be plugged into a cell membrane (**Fig. 2**). Indeed, a model element, once defined in NeuroML, acts as a building block that may be reused any number of times within or across models. Such reuse of model component speeds model construction and prototyping irrespective of the simulation engine used.

The defined structure of each model element and the relationships between them inform users of exactly how model elements are to be created and combined. This encourages the construction of well-structured models, reduces errors and redundancy and ensures that FAIR principles are firmly baked into NeuroML models and the ecosystem of tools. As we will see in the following sections, NeuroML's formal structure also enables features such as model validation prior to simulation, translation into simulation specific formats, and the use of NeuroML as a common language of exchange between different tools.

## NeuroML supports a large ecosystem of software tools that cover all stages of the model life cycle

Model building and the generation of scientific knowledge from simulation and analysis of models is a multi-step, iterative process requiring an array of software tools. NeuroML supports all stages of the model development life cycle (**Fig. 1**), by providing a single model description format that interacts with a myriad of tools throughout the process. Researchers typically assemble ad-hoc sets of scripts, applications, and processes to help them in their investigations. In the absence of standardization, they must work with the specific model formats and APIs that each tool they use requires, and somehow convert model descriptions when using multiple applications in a toolchain. NeuroML addresses this issue by providing a common language for the use and exchange of models and their components between different simulation engines and modeling tools. The NeuroML ecosystem includes a large collection of software tools, both developed and maintained by the main NeuroML contributors (the "core NeuroML tools") and those external applications which have added NeuroML support (**Fig. 3**, **Appendix 1 Tables 1** and **2**).

| Link | Description | Model life cycle stages |
|---|---|---|
| Guide 1 | Create and simulate a simple regular spiking Izhikevich neuron in NeuroML | Create, Validate, Simulate |
| Guide 2 | Create a network of two synaptically connected populations of Izhikevich neurons | Create, Validate, Visualize, Simulate |
| Guide 3 | Build and simulate a single compartment Hodgkin-Huxley neuron | Create, Validate, Visualize, Simulate |
| Guide 4 | Create and simulate a multi compartment hippocampal OLM neuron | Create, Validate, Visualize, Simulate |
| Guide 5 | Optimize/fit NeuroML models to experimental data | Create, Validate, Simulate, Fit |
| Guide 6 | Guide to converting cell models to NeuroML and sharing them on Open Source Brain | Create, Validate, Simulate, Share |
| Guide 7 | Create novel NeuroML models from components on NeuroML-DB | Reuse, Create, Validate, Simulate |
| Guide 8 | Extend NeuroML by creating a novel model type in LEMS | Create, Simulate |

**Table 1.**

Step by step guides for using NeuroML, illustrating the various stages of the model development lifecycle. An updated list is available at *http://neuroml.org/gettingstarted*.

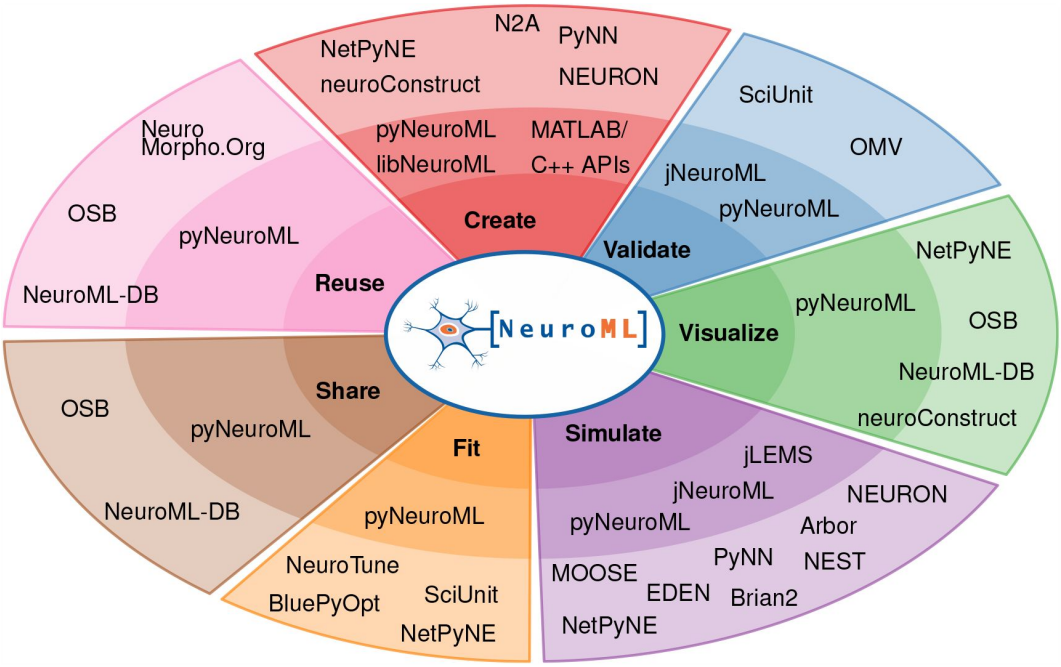



**Figure 3.**

The ecosystem of NeuroML complaint tools and their relation to the model life cycle. The inner circle shows the core NeuroML tools that are maintained directly by the NeuroML developers. These provide core functionality to read, modify, or create new NeuroML models, as well as to analyze, visualize and simulate the models. The outermost layer shows NeuroML compliant tools which have been developed independently to allow various interactions with NeuroML models. These complement the core tools by facilitating model creation, validation, visualization, simulation, fitting/optimization, sharing and reuse. Further information on each of the tools shown here can be found in Appendix 1 Tables 1 and 2.

The core NeuroML tools currently include APIs in several programming languages—Python, Java, C++ and MATLAB. These tools provide critical functionality to allow users to interact with NeuroML components and build models. Using these, researchers can build models from scratch, or read, modify, analyze, visualize, and simulate existing NeuroML models on supported simulation platforms. The simulation platforms (e.g. EDEN (***Panagiotou et al., 2022***), NEURON (***Hines and Carnevale, 1997***)), along with other independently developed tools, form the next layer of the software ecosystem—providing extra functionality such as interactive model construction (e.g. neuroConstruct (***Gleeson et al., 2007***), NetPyNE (***Dura-Bernal et al., 2019***)), additional visualization (e.g. OSB (***Gleeson et al., 2019***)), analysis (e.g. NeuroML-DB (***Birgiolas et al., 2023***)), data-driven validation (e.g. SciUnit (***Gerkin et al., 2019***)), and archival/sharing (e.g. OSB, NeuroML-DB). Indeed, OSB and NeuroML-DB are prime examples of how advanced neuroinformatics resources can be built on top of standards such as NeuroML.

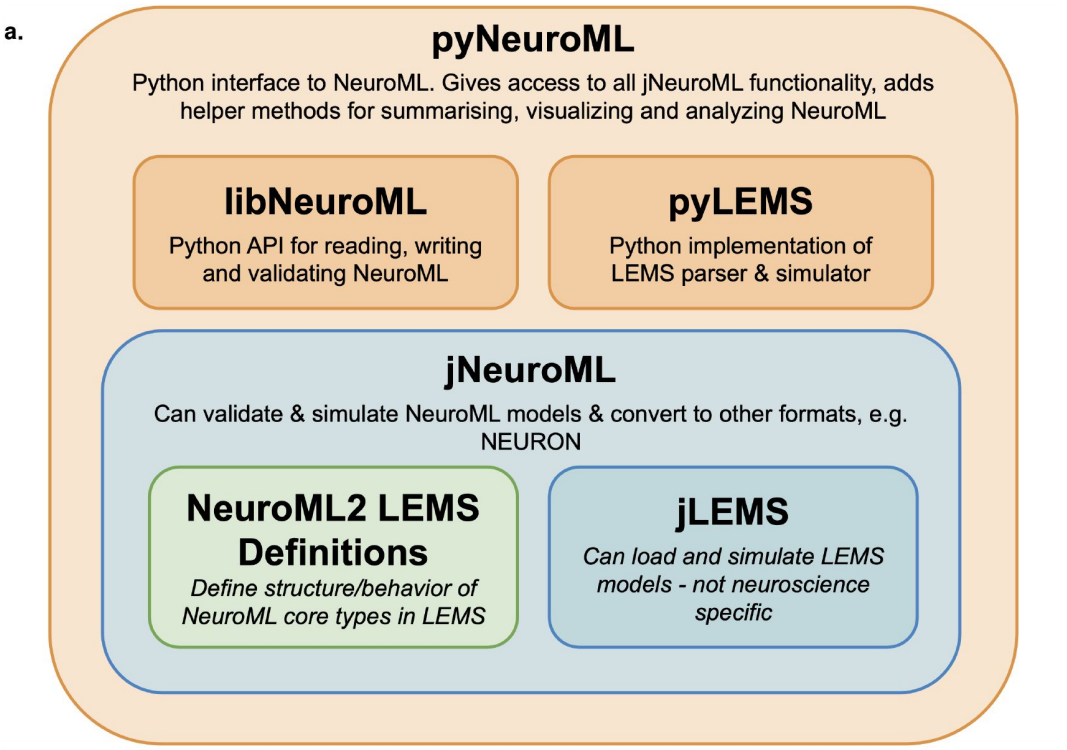
**Table 1** lists a number of interactive, step-by-step guides in the NeuroML documentation, illustrating how software tools can be used to achieve specific tasks across the model development life cycle. In the following sections we discuss the specific functionality available at each stage of model development.

## Creating NeuroML models

The structured declarative elements of NeuroMLv2, when combined with a procedural scripting language such as Python, provide a powerful and yet intuitive "building block" approach to model construction. For this reason, Python is now the recommended language for interacting with NeuroML (**Fig. 4**), although XML remains the primary serialization language for the format (i.e. for saving to disk and depositing in model repositories (**Fig. 5**)). Python has emerged as a key programming language in science, including many areas of neuroscience (***Muller et al., 2015***). A Python based NeuroML ecosystem ensures that users can take advantage of Python's features, and also use packages from the wider Python ecosystem in their work (e.g. Numpy (***Harris et al., 2020***), Matplotlib (***Hunter, 2007***)). pyNeuroML, the Python interface for working with NeuroML, is built on top of the Python NeuroML API, libNeuroML (***Vella et al., 2014***; ***Sinha et al., 2023a***,b) (**Fig. 4**).

As illustrated in **Fig. 5**, Python can be used to combine different NeuroML components into a model. NeuroML supports a number of pathways for the creation of new models. Modelers may use elements included in the NeuroML standard, re-use user-defined NeuroML model elements from other models or define completely new model elements using LEMS (**Fig. 5**) (see section on extending NeuroML below). It is common for models to use a combination of these strategies, e.g. ***Gurnani and Silver (2021***); ***Kriener et al. (2022***); ***Cayco-Gajic et al. (2017***), highlighting the flexibility provided by the modular design of NeuroML. NeuroML APIs support all these workflows. The Python tools also include many additional higher level utilities to speed up model construction, such as factory functions, type hints, and convenience functions for building complex multicompartmental neuron models (Box 1).

For construction of complex 3D circuit models, or for users who are not experienced with Python, there are a number of NeuroML compliant online and standalone applications with graphical user interfaces for model construction. These include NetPyNE's interactive web interface (***Dura-Bernal et al., 2019***) (which is available on the latest version of OSB[3]) and neuroConstruct (***Gleeson et al., 2007***) which can export models directly into NeuroML and LEMS. These applications can be used to build and simulate new NeuroML models without requiring programming. Thus, users can take advantage of the individual features provided by these applications to generate NeuroML compliant models and model elements.




**b.** Example Python usage

```python
from neuroml import * # NeuroML API libNeuroML

newdoc = NeuroMLDocument(id="new_doc")
newcell = IafTauCell(id="cell_0",
    leak_reversal="-60mV", thresh="0mV",
    tau="5ms", reset="-70mV")
newdoc.add(newcell)

network = newdoc.add(Network, id="new_net",
    validate=False)
population = network.add(Population,
    id="new_pop", size=10,
    component=newcell.id)

# Helper method to ensure all parameters
# present and appropriate
newdoc.validate(recursive=True)
```

**c.** XML serialization

```xml
<neuroml id="new_doc">

  <iafTauCell id="cell_0" leakReversal="-60mV"
    thresh="0mV" reset="-70mV" tau="5ms"/>

  <network id="new_net">

      <population id="new_pop"
    component="cell_0" size="10"/>

  </network>

</neuroml>
```

## Figure 4.

The core NeuroML software stack, and an example NeuroML model created using the Python NeuroML tools. **(a)** The core NeuroML software stack consists of Java (blue) and Python (orange) based applications/libraries, and the LEMS model ComponentType definitions (green), wrapped up in a single package, pyNeuroML. Each of these modules can be used independently or the whole stack can be obtained by installing pyNeuroML with the default Python package manager, Pip: pip install pyneuroml. **(b)** An example of how to create a simple NeuroML model is shown, using the NeuroMLv2 Python API (libNeuroML) to describe a model consisting of a population of 10 integrate and fire point neurons (IafTauCell) in a network. The IafTauCell, Network, Population, and NeuroMLDocument model ComponentTypes are provided by the NeuroMLv2 standard. The underlying dynamics of the model are hidden from the user, being specified in the LEMS ComponentType definitions of the elements (see Methods). The simulator-independent NeuroML model description can be simulated on any of the supported simulation back-ends. **(c)** XML serialization of the NeuroMLv2 model description shows the correspondence between the Python object model and the XML serialization.

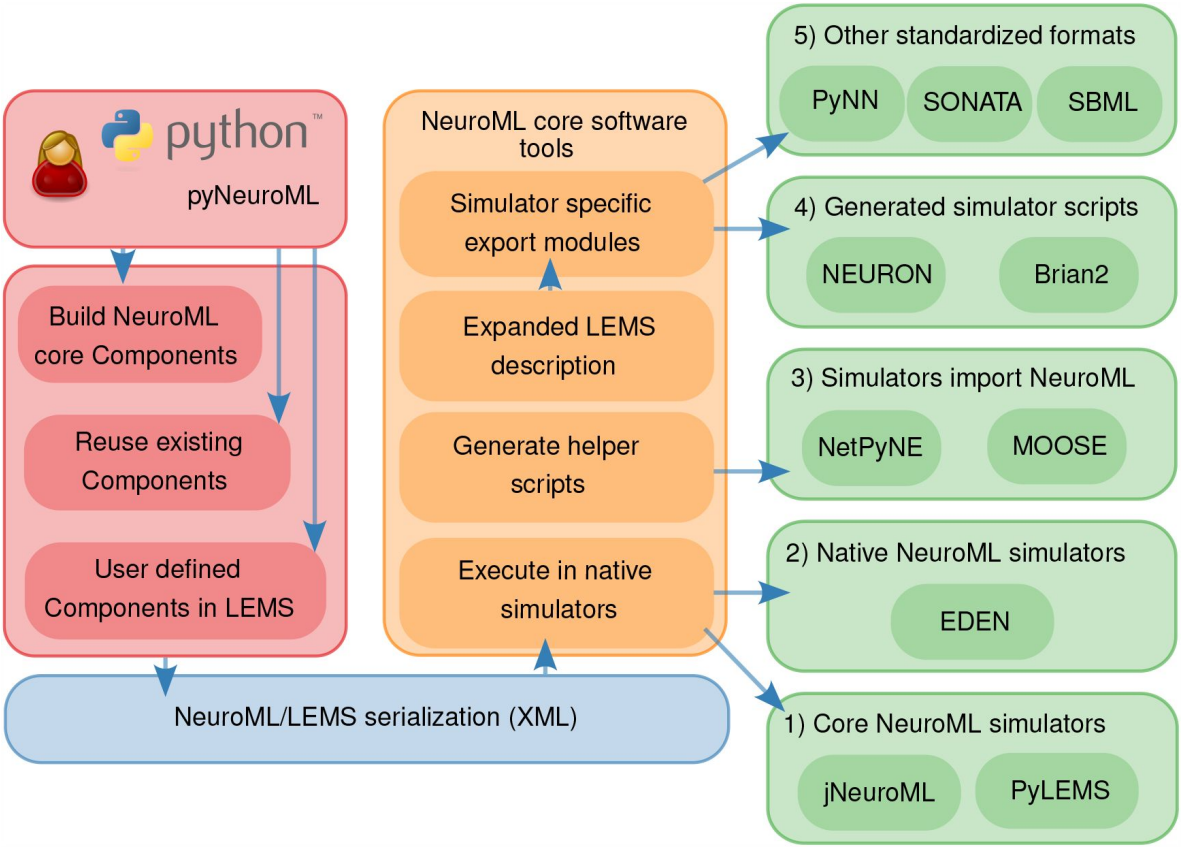



**Figure 5.**

Workflow showing how to create and simulate NeuroML models using Python. The Python API can be used to create models which can include elements built from scratch from the core NeuroML standard, re-use elements from previously created models, or create new components based on novel model definitions expressed in LEMS (red). The generated model elements are saved in the default XML-based serialization (blue). The NeuroML core tools (orange) include modules to import model descriptions expressed in the XML serialization, and supports multiple options for how simulators can execute these models (green). These include native execution of the NeuroML models by core simulators (1) and others such as EDEN (2) and generation of helper Python scripts allowing direct importation of the NeuroML model by simulators which support this (3). For other simulators, the fully expanded LEMS description of the models can be generated, which can be mapped to simulator specific formats to generate scripts to run in the target simulators (4). Finally, a number of other standardized formats are supported (5), to which the loaded model can be mapped.

## Validating NeuroML models

Ensuring a model is "valid" can have different meanings at different stages of the development, testing and analysis of models - from checking whether the source files are in the correct format, to ensuring the model reproduces a significant feature of its biological counterpart. NeuroML's hierarchical, well-defined structure allows users to check their model descriptions for correctness at multiple levels (**Fig. 6**), in a manner similar to multi-level testing in software development. Importantly, a majority of the validation tests in NeuroML are run on the models' NeuroML descriptions *prior to simulation*.

A first level of validation checks the structure of individual model elements against their formal specifications contained in the NeuroML standard. The standard includes information on the parameters of each model element, restrictions on parameter values, their allowed units, their cardinality, and the location of the model element in the model hierarchy—i.e. parent/children relationships. A second level of validation includes a suite of semantic and logical checks. For example, at this level, a model of a multicompartmental cell can be checked to ensure that all segments referenced in segment groups (e.g. the group of dendritic segments) have been defined, and only defined once with unique identifiers. A list of validation tests currently included in the NeuroML core tools can be found in Appendix 1 Table 3. These can be run against NeuroML files at the command line or programmatically in Python—Box 1.

A key advantage of using the NeuroML2/LEMS framework is that dimensions and units are inbuilt into LEMS descriptions. This enables automated conversions of units, unit checking, together with the validation of equations. Any expressions in models which are dimensionally inconsistent will be highlighted at this stage. Note that LEMS handles unit conversions "under the hood"—modelers have flexibility in how they enter the *units* of parameter values (e.g. specifying conductance density in $S/m^2$ or $mS/cm^2$) in the NeuroML files, with the underlying LEMS definitions ensuring that a consistent set of *dimensions* are used in model equations (***Cannon et al., 2014***). LEMS then takes care of mapping the entered units to the target simulator's preferred units. This makes model definition, inspection, use, extension, and translation easier and less error-prone.

Once the set of NeuroML files are validated, the model can be simulated, and checks can be made to test whether execution produces consistent results (e.g. firing rate of neurons in a given population) across multiple simulators (or versions of the same simulator). For this, the OSB Model Validation (OMV) framework has been developed (***Gleeson et al., 2019***). This framework can automatically check that the output (e.g. spike times) of a NeuroML model running on a given simulator is within an allowed tolerance of the expected value. OMV has been applied to NeuroML models that have been shared on OSB, to test consistent behavior of models as the models themselves, and all supported simulators, are updated. This has proven to be a valuable ongoing process for ensuring uniform usage and interpretation of NeuroML across the ecosystem of supporting tools.

A final level of validation concerns checking whether the model elements have emergent features that are in line with experimentally observed behavior of the biological equivalents. NeuronUnit (***Gerkin et al., 2019***), a SciUnit (***Omar et al., 2014***) package for data-driven unit testing and validation of neuronal and ion channel models, is also fully NeuroML compliant, and also supports automated validation of NeuroML models shared on NeuroML-DB and OSB.

## Visualizing/analyzing NeuroML models

Multiple visualization, inspection, and analysis tools are available in the NeuroML software ecosystem. Since NeuroML models have a fixed, well defined structure, the core NeuroML libraries can extract all information from their descriptions. This information can be used by modelers and their programs/tools to run automated programmatic analyses on models.

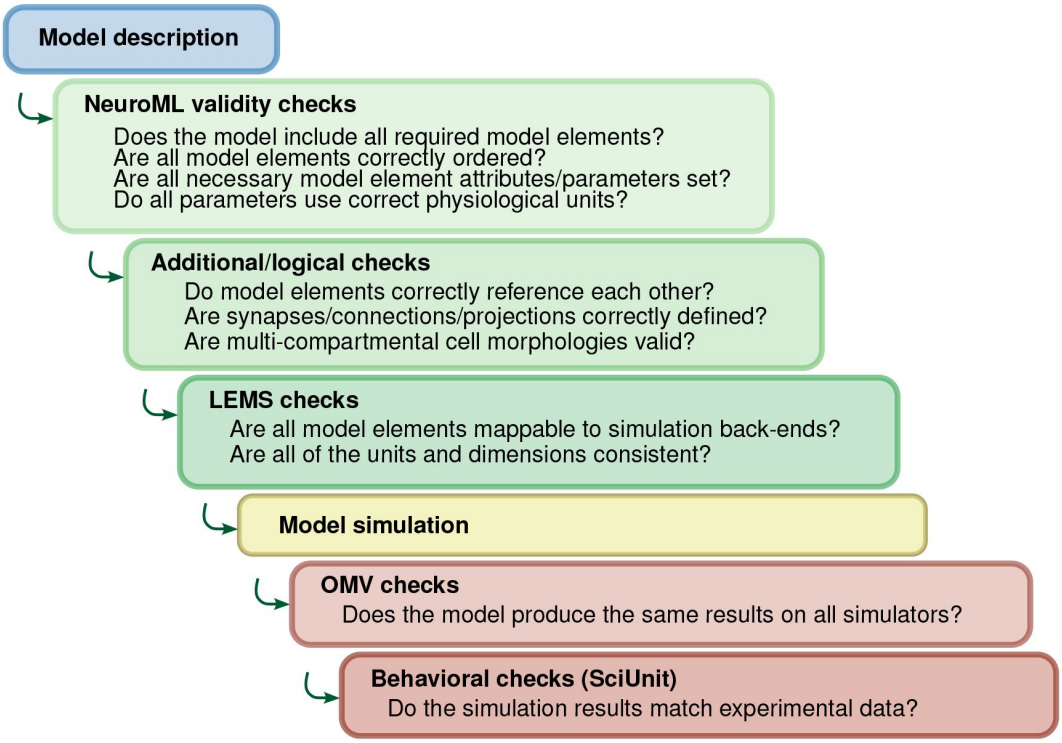
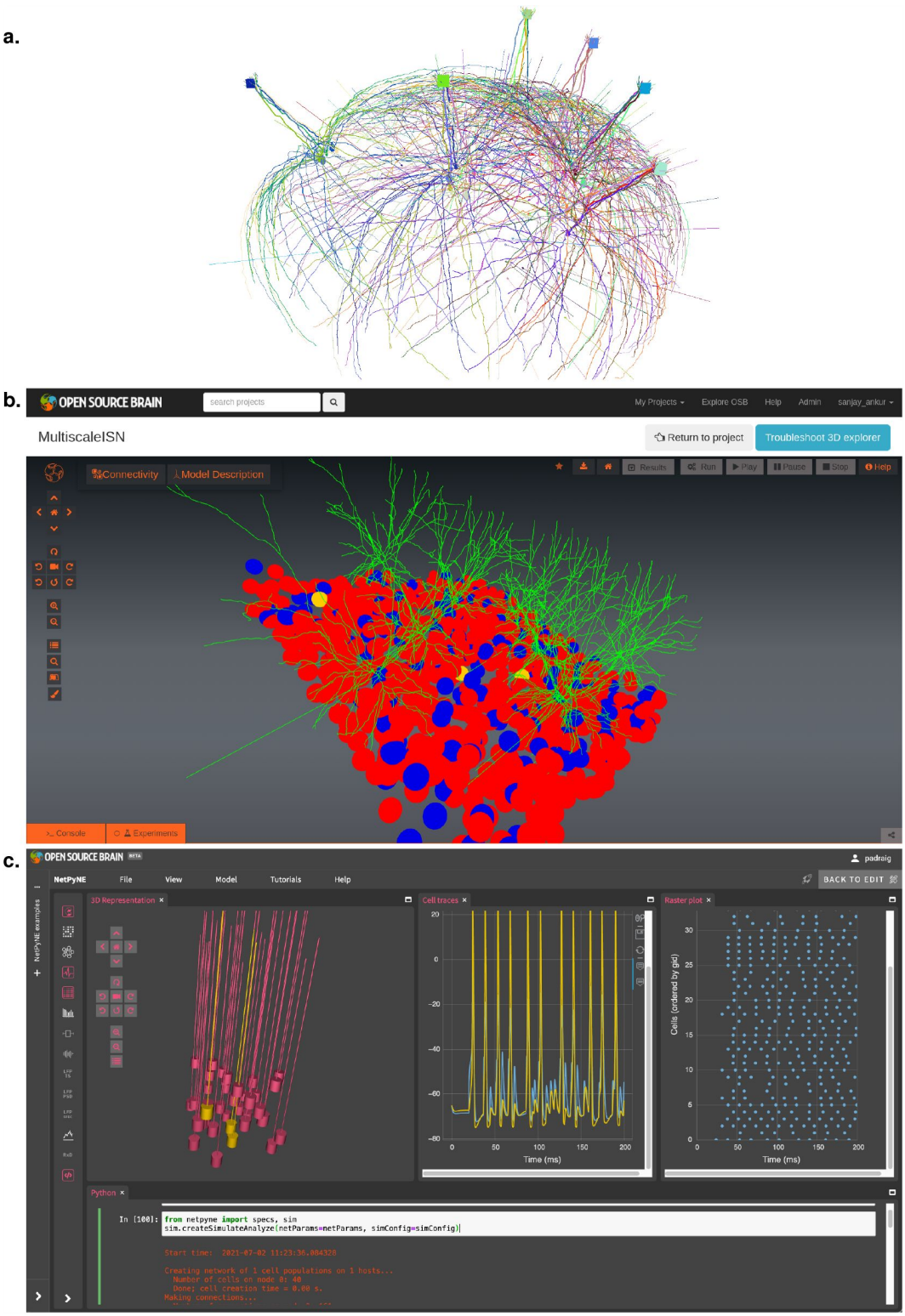



**Figure 6.**

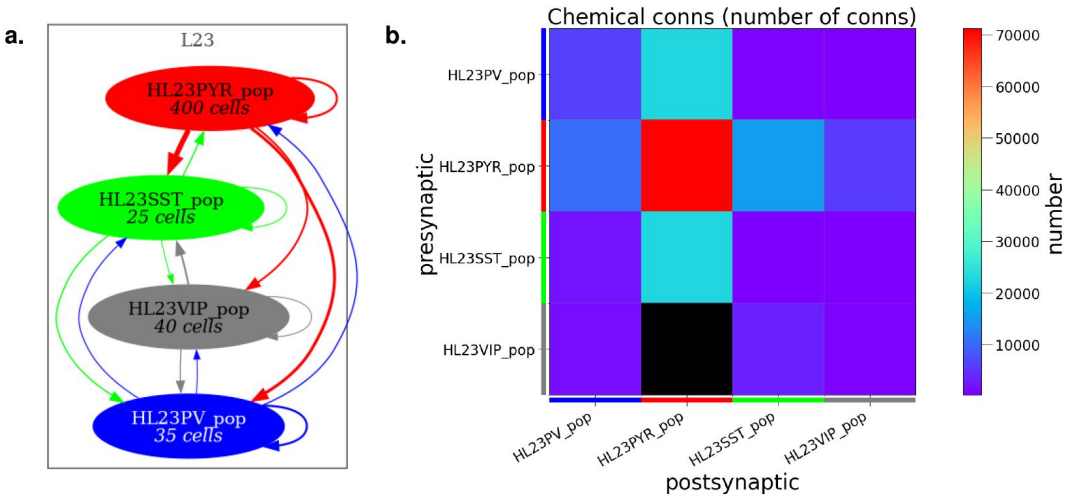
NeuroML model development incorporates multi-level validation of models. Checks are performed on the model descriptions (blue) before simulation using validation at both the NeuroML and LEMS levels (green). After the models are simulated (yellow), further checks can be run to ensure the output is in line with expected behavior (brown). The OSB Model Validation (OMV) framework can be used to ensure consistent behavior across simulators, and comparisons can be made of model activity to their biological equivalents using SciUnit.

pyNeuroML includes a range of ready-made inspection utilities for users (Box 1) that can be used via Python scripts, interactive Jupyter Notebooks, and command line tools. Examining the structure of cell and network models with 2D and 3D views is important for manual validation and to compare them to their biological counterparts. Graphical views of cell model morphology and the 3-dimensional network layout (**Fig. 7**), population and connectivity matrices/graphs at different levels (**Fig. 8**), and model summaries including the mathematical dynamics can all be generated (**Fig. 9**). In addition to these inspection functions, a number of utilities for the inspection of NeuroML descriptions of electrophysiological properties of membrane conductances and their spatial distribution over the neuronal membrane are also provided (**Fig. 9**).



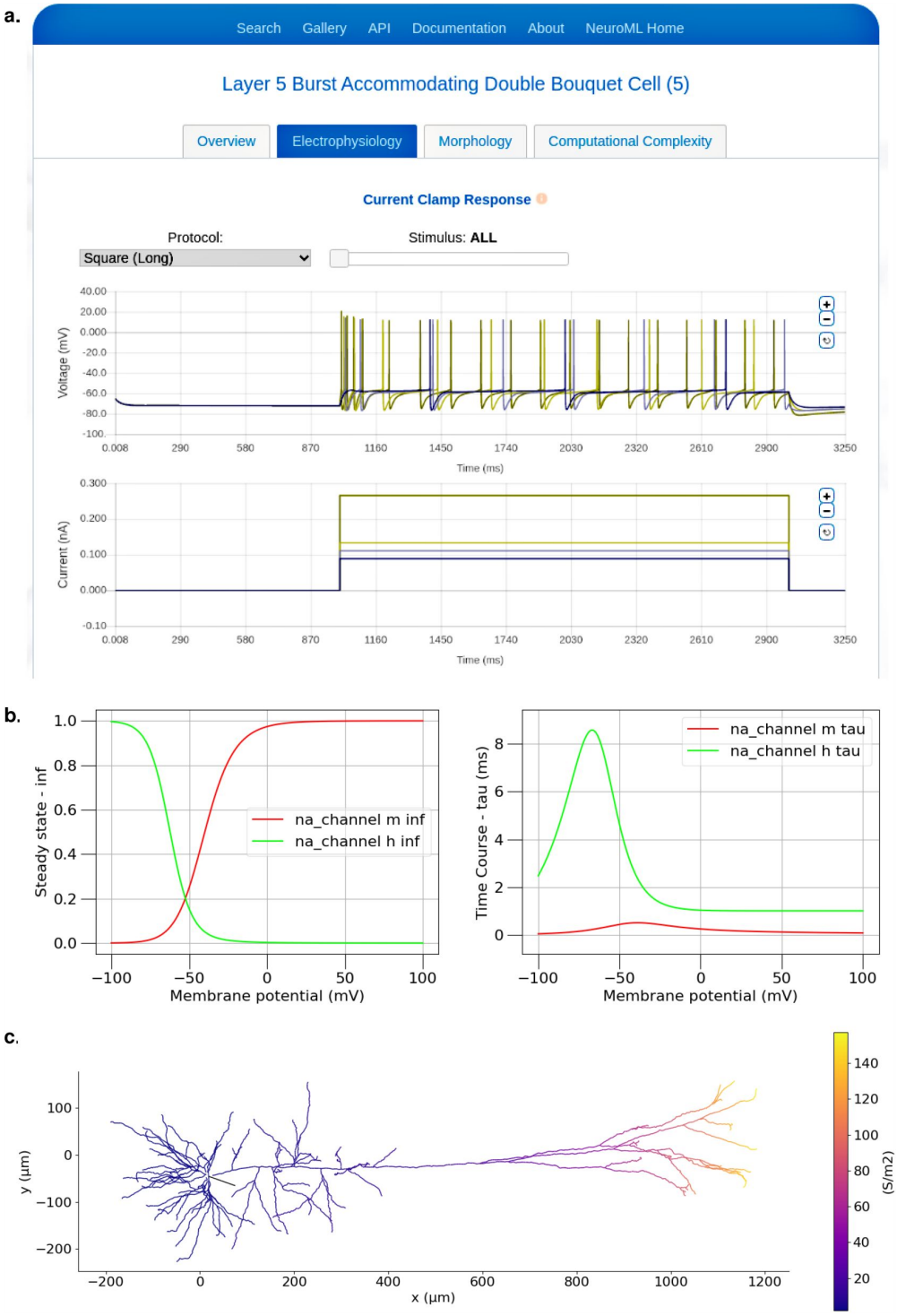

## Figure 7.

Visualization of detailed neuronal morphology of neurons and networks together with their functional properties (results from model simulation) enabled by NeuroML. **(a)** interactive 3-D (VisPy (***Campagnola et al., 2023***) based) visualization of an olfactory bulb network with detailed mitral and granule cells (***Migliore et al., 2014***), generated using pyNeuroML. **(b)** Visualization of an inhibition stabilized network based on ***Sadeh et al***. (***2017***) using Open Source Brain (OSB) version 1 (***Gleeson et al., 2019***). **(c)** Visualization of network of simplified multicompartmental neurons together with spiking properties using NetPyNE's GUI (***Dura-Bernal et al., 2019***), which is embedded in OSB version 2.




## Figure 8.

Analysis and visualization of network connectivity from NeuroML model descriptions *prior to simulation*. Network connectivity schematic **(a)** and connectivity matrix **(b)** for a half scale implementation of the human layer 2/3 cortical network model (***Yao et al., 2022***) generated using pyNeuroML.




## Figure 9.

Examples of analysis functions available for a NeuroML model neuron. **(a)** Electrophysiological analysis features provided by the NeuroML-DB web based platform (***Birgiolas et al., 2023***). Plots show four superimposed voltage traces in the top panel and corresponding current injection traces below). **(b)** Example plots of steady states of activation (na_channel na_m inf) and inactivation (na_channel na_h inf) variables and their time courses (na_channel na_m tau and na_channel na_h tau) for the Na channel from the classic Hodgkin Huxley model (***Hodgkin and Huxley, 1952***). **(c)** The distribution of the peak conductances for the Ih channel over a layer 5 Pyramidal cell (***Hay et al., 2011***). Both (b) and (c) were generated using the analysis features in pyNeuroML, and similar functionality is also available in OSB (***Gleeson et al., 2019***).

**Box 1.**

## NeuroML Python tools for users

PyNeuroML, based on the Python libNeuroML API, provides Python functions and command line utilities that support all stages of the model life cycle.

**Create**

```
# Create a container document
doc = NeuroMLDocument(id="network0")

# Add single exponential synapse model
syn0 = doc.add("ExpOneSynapse", id="syn0", gbase="65nS", erev="0mV", tau_decay="3ms")

# Reuse existing ion channel model
doc.add("IncludeType", href="Na_chan.channel.nml")

# Create a cell with 3D morphology using the Cell ComponentType
cell = doc.add("Cell", id="olm", neuro_lex_id="NLXCELL:091206") # Hippocampal CA1 OLM cell
cell.set_init_memb_potential("-67mV")
cell.set_resistivity("0.15 kohm_cm")
cell.add_channel_density(doc, cd_id="na_all", cond_density="10 mS_per_cm2",
                         ion_channel="Na_chan", ion_chan_def_file="Na.channel.nml",
                         erev="50mV", ion="na")
cell.add_unbranched_segment_group("soma_group")
soma_0 = cell.add_segment(prox=[0, 0, 0, 10], dist=[0, 10, 0, 10], name="Seg0_soma_0",
                          group_id="soma_group", seg_type="soma")
...
```

**Validate**

```
validate_neuroml2("file.nml")                $ pynml "file.nml" -validate
doc.validate(recursive=True)
```

**Inspect and visualize**

```
element.info()
summary(doc)                                 $ pynml-summary "file.nml"
nml2_to_png(doc)                             $ pynml -png "file.nml"
nml2_to_svg(doc)                             $ pynml -svg "file.nml"
generate_nmlgraph(doc)                       $ pynml "file.nml" -graph
                                             $ pynml "file.nml" -matrix 1
plot_2D(cell)                                $ pynml-plotmorph "cell.nml"
plot_interactive_3d(cell)                    $ pynml-plotmorph -interactive3d "cell.nml"
plot_interactive_3d(network)                 $ pynml-plotmorph -interactive3d "net.nml"
                                             $ pynml-channelanalysis "channel.nml"
plot_channel_densities(cell)                 $ pynml-plotchan "cell.nml"
```

**Simulate**

```
run_lems_with_jneuroml("sim.xml")            $ pynml "sim.xml"
run_lems_with_jneuroml_neuron("sim.xml")     $ pynml "sim.xml" -neuron -run
run_lems_with_jneuroml_netpyne("sim.xml")    $ pynml "sim.xml" -netpyne -run
run_on_nsg("jneuroml_neuron", "sim.xml")
...
```

**Share and archive**

```
create_combine_archive("sim.xml")            $ pynml-archive "neuron.cell.nml"
```

The graphical applications included in the NeuroML ecosystem (e.g. neuroConstruct, NeuroML-DB, OSB (v1[4] and v2), NetPyNE, and Arbor-GUI) also provide many of their own analysis and visualization functions. OSBv1, for example, supports automated 3D visualization of networks and cell morphologies, network connectivity graphs and metrics, and advanced model inspection features (***Gleeson et al., 2019***) (**Fig. 7b**). On OSBv2, NetPyNE provides advanced graphical plotting and analysis facilities (**Fig. 7c**). A complete JupyterLab[5] interface is also included in OSBv2 for Python scripting, allowing interactive notebooks to be created and shared, mixing scripting and graphical elements, including those generated by pyNeuroML. NeuroML-DB also provides information on electrophysiology, morphology, and the simulation aspects of neuronal models, such as their computational complexity, i.e. an indication of how "fast" the model will run (***Birgiolas et al., 2023***) (**Fig. 9a**). In general, any NeuroML compliant application can be used to inspect and analyze elements of NeuroML models, each having their own distinct advantages.

### Simulating NeuroML models

Users can simulate NeuroML models using a number of simulation "back ends" without making any changes to their models. This is because the NeuroML/LEMS descriptions of the models can be automatically translated into any number of simulator specific formats. pyNeuroML facilitates access to all available simulation options, both from the command line and using single function calls in Python scripts when using the API (Box 1).

The simulation back ends can be classified into five broad categories (**Fig. 5**):

1. core NeuroML/LEMS simulation engines (reference implementations)
2. other simulators that natively support NeuroML
3. simulators that import/translate NeuroML to their own internal formats
4. back ends which are supported through generation of simulator-specific scripts by the core NeuroML tools.
5. export to other standardized formats which may allow simulation/analysis in other packages.

Having multiple strategies in place for supporting NeuroML gives more freedom to simulator developers to choose how much they wish to be involved with implementing and supporting NeuroML functionality in their applications, while maximizing the options available for end users.

The first category includes jLEMS, jNeuroML, and PyLEMS (**Fig. 4**). jLEMS serves as the reference implementation for the LEMS language, and as such it can simulate any model described in LEMS (not necessarily from neuroscience). When coupled with the LEMS definitions of NeuroML standard entity structure/dynamics, it can simulate most NeuroML models, though it does not currently support numerically solving the cable equation (***Rall, 1962***), which is required for multicompartmental neurons. jNeuroML bundles the NeuroML standard LEMS

definitions, jLEMS and other functionality into a single package for ease of installation/usage. There is also a pure Python implementation of a LEMS interpreter, PyLEMS, which can be used in a similar way to jLEMS. The pyNeuroML package encapsulates all of these tools to give easy access (at both command line and in Python) to all of their functionality (Box 1).

The second category deals with simulators which use NeuroML natively as their base descriptions of modeling entities, as opposed to having their own specific formats. The EDEN simulator is one such recently developed example, which was designed from its inception to read NeuroML and LEMS models for efficient, parallel simulation (***Panagiotou et al., 2022***).

The third category involves simulators which have their own internal formats, and include methods to import NeuroMLv2/LEMS and in this way translate the models to their own formats. Examples include NetPyNE (***Dura-Bernal et al., 2019***), MOOSE (***Ray and Bhalla, 2008***), N2A (***Rothganger et al., 2014***).

The fourth category comprises simulators for which scripts can be generated by pyNeuroML. These include native NEURON code (consisting of Python, and the simulator's hoc and NMODL format files), the Arbor simulator (which reuses many NEURON formats, e.g. NMODL for ionic conductances and hoc for morphologies), and the Brian simulator (***Stimberg et al., 2019***).

The final category consists of export options to standardized formats in neuroscience and the wider computational biology field, which enable interaction with simulators and applications supporting those formats. These include the PyNN package (***Davison et al., 2009***), which can be run in either NEURON, NEST (***Gewaltig and Diesmann, 2007***) or Brian, the SONATA data format (***Dai et al., 2020***) and the SBML standard (***Hucka et al., 2003***) (see Reusing NeuroML models for more details).

Each of these simulation options have different strengths and limitations, and NeuroML users can take advantage of these by simply changing their simulation back end when simulating their model. Many of these tools also support exporting NeuroML models once they have been created. It is important to note though that not all NeuroML models can be exported to/are supported by each of these target simulators. This depends on the capabilities of the simulator in question (whether it supports networks, or morphologically detailed cells) and pyNeuroML will provide feedback if a feature of the model is not supported in a chosen environment.

A Python based ecosystem ensures that automated simulation of models can easily be carried out either using scripts, or the command line tools. Utilities to facilitate the execution of simulations on dedicated supercomputing resources, such as the Neuroscience Gateway (NSG)[6] (***Sivagnanam et al., 2013***) are also available within the ecosystem. OSBv1 takes advantage of these to support the submission of NeuroML model simulation jobs using the NEURON simulator on NSG. NetPyNE also includes batch processing and parameter exploration features, and its deployments on the cloud based OSBv2 platform supports on-demand scaling of computing resources. Finally, the JupyterLab environment on OSBv2 contains all of the core NeuroML tools and various simulation back ends as pre-installed software packages, ready to use.

### Optimizing NeuroML models

Development of biologically detailed models of brain function requires that components and emergent properties match their behavior of the corresponding biology as closely as possible. Thus, fitting neurons and networks to experimental data is a critical step in the model life cycle (***Rossant et al., 2011***; ***Druckmann et al., 2007***). NeuroML promotes data-driven modeling by providing functions to fit and optimize NeuroML models against experimental data in pyNeuroML. pyNeuroML includes the NeuroMLTuner module[7], which builds on the Neurotune package[8] for tuning and optimizing NeuroML models against data using evolutionary computation techniques. This module allows users to select a set of weighted features from their

data to calculate the fitness of populations of candidate models. In each generation, the fittest models are found and mutated to create the next generation of models, until a set of models that best exhibit the selected data features are isolated (see Guide 5 in **Table 1**)[9].

The NeuroML ecosystem includes a number of tools that also provide model fitting features. The Blue Brain Python Optimisation Library (BluePyOpt) (***Van Geit et al., 2016***), an extensible framework for data-driven model parameter optimization, supports exporting optimized models to NeuroML files[10]. Similar to pyNeuroML, NetPyNE also uses the inspyred Python package[11] to provide evolutionary computation based model optimization features (***Dura-Bernal et al., 2019***).

### Sharing NeuroML models

The NeuroML ecosystem includes the advanced web based model sharing platforms NeuroML-DB (***Birgiolas et al., 2023***)[12] and OSB (***Gleeson et al., 2019***). These resources have been designed specifically for the dissemination of models and model elements standardized in NeuroML. The integrated collaborative OSB platform also supports visualization, analysis, simulation, and development of NeuroML models. Researchers can create shared, collaborative NeuroML projects on them and can take advantage of the in-built automated visualization and analysis pipelines to explore and re-use models and their components. Whereas version 1 (OSBv1) focused on providing an interactive 3D interface for running pre-existing NeuroML models (e.g. sourced from linked GitHub repositories) (***Gleeson et al., 2019***), OSBv2 provides cloud based workspaces for researchers to construct NeuroML based computational models as well as analyze, and compare them to, the experimental data on which they are based, thus facilitating data-driven computational modeling. Appendix 1 Table 6 provides a list of stable, well tested NeuroML compliant models from brain regions including the neocortex, cerebellum and hippocampus, which have been shared on OSB.

NeuroML-DB aims to promote the uptake of standardized NeuroML models by providing a convenient location for archiving and exploration. It includes advanced database search functions, including ontology based search (***Birgiolas et al., 2015***), coupled with pre-computed analyses on models' electrophysiological and morphological properties, as well as their computational complexity (and indication of the relative speed of execution of different models).

NeuroML's modular nature ensures that models and their components can be easily shared with others through standard code sharing resources. The simplest way of sharing NeuroML models and components is to share their Python descriptions or their XML serializations available through these resources. Indeed, it is straightforward to make Python descriptions or the XML serializations available via different file, code (GitHub, GitLab), model sharing (ModelDB (***Migliore et al., 2003***; ***McDougal et al., 2017***)), and archival (Zenodo, Open Science Framework) platforms, just like any other code/data produced in scientific investigations. Complex models with many components, spanning multiple files, such as networks and neuronal models that reference multiple cell and ionic conductance definitions, can also be exported into a COMBINE archive (***Bergmann et al., 2014***), a zip file with a manifest that includes metadata about its contents. pyNeuroML includes functions to easily create COMBINE archives from NeuroML models and simulations (Box 1).

OSB is designed so that researchers can share their code on their chosen platform (e.g. GitHub), while retaining full control over write access to their repositories. Afterwards, a page for the model can be created on OSB which lists the latest files present there, with links to OSB visualization/analysis/simulation features which can use the standardized files found in the resource.

NeuroML supports the embedding of structured ontological information in model descriptions (***Neal et al., 2018***). Models can include NeuroLex (now InterLex) (***Larson and Martone, 2013***) identifiers for their components (e.g. neuro_lex_id in Box 1). This links model components to their biological counterparts and makes them more transparent, findable, and reusable. For example, different types of neurons and brain regions have unique ontological ids. A user can use these ids to search for relevant model components on NeuroML-DB.

### Reusing NeuroML models

NeuroML models, once openly shared, become community resources that are accessible to all. Researchers can use models shared on NeuroML-DB and OSB without restrictions. Guide 7 in **Table 1** provides an example of finding NeuroML based model components using the API of NeuroML-DB, and creating novel models incorporating these elements.

In addition to these platforms, other experimental data and model dissemination platforms also provide standardized NeuroML versions of relevant models to promote uptake and reuse. For example, Neuro*Morpho.org* (***Ascoli et al., 2007***) includes a tool to download NeuroML compliant versions of its cellular reconstructions. NeuroML versions of models released by organizations such as the Blue Brain Project (***Markram et al., 2015***) (whole cell models as well as ion channel models from Channelpedia (***Ranjan et al., 2011***)), the Allen Institute for Brain Science (***Billeh et al., 2020***), and the OpenWorm project (***Gleeson et al., 2018***) are also openly available for reuse (Appendix 1 Table 6).

NeuroML can also interact with other standards to further promote model re-use. Whereas NeuroML is a declarative standard, PyNN (***Davison et al., 2009***) is a procedural standard with a Python API for creating network models that can be simulated on multiple simulator back-ends. NeuroML models which are within the scope of PyNN can be converted to the PyNN format, and vice-versa. Similarly, NeuroML also interacts with SONATA (***Dai et al., 2020***) data format by supporting the two way conversion of the network structures of NeuroML models into SONATA. In standards not specific to neuroscience, models from the well established SBML standard (***Hucka et al., 2003***) can be converted to LEMS (***Cannon et al., 2014***), for inclusion in neuroscience related modeling pipelines, and a subset of NeuroML/LEMS models can be exported to SBML, which allows use with simulators and analysis packages compliant to this standard, e.g. Tellurium (***Choi et al., 2018***). Simulation execution details of NeuroML/LEMS models can also be exported to Simulation Experiment Description Markup Language (SED-ML) (***Waltemath et al., 2011***), allowing advanced resources such as Biosimulators[13] (***Shaikh et al., 2022***) to feature NeuroML models.

## NeuroML is extensible

While the core NeuroML elements (Appendix 1 Tables 4 and 5) provide a broad range of curated model types for simulation based investigations, NeuroML can be extended (using LEMS) to incorporate novel model elements and types when they are not (yet) available in the core standard.

LEMS is a general purpose model specification language for creating fully machine readable definitions of the structure and behavior of model elements (***Cannon et al., 2014***). The dynamics of NeuroML elements are described in LEMS. The hierarchical nature of LEMS means that new elements can build on pre-existing elements of the modular NeuroML framework. For example, a novel ionic conductance element can extend the “ionChannelHH” element, which in turn extends “baseIonChannel”. Thus, the new element will be known to the NeuroML elements as depending on an external voltage and producing a conductance, properties that are inherited from “baseIonChannel”. Other elements, e.g. a cell, can incorporate this new type without needing any other information about its internal workings.

LEMS (and therefore NeuroML) element definitions (called “ComponentTypes”) specify the dynamical behavior of the model element in terms of a list of yet to be set parameters. Once the generic model behavior is defined, modelers only need to fill in the appropriate values of the required parameters (e.g. conductance density, reversal potential, etc.) to create new instances (called “Components”) of the element (see Methods for more details). Users can therefore create arbitrary, reusable model elements in LEMS, which can be treated the same way as core model elements (for an example see Guide 8 in **Table 1**).

Another major advantage of NeuroML’s use of the LEMS language is its translatability. Since LEMS is fully machine readable, its primitives (e.g. state variables and their dynamics, expressed as ordinary differential equations) can be readily mapped into other languages. As a result, simulator specific code (***Blundell et al., 2018***) can be generated from NeuroML models and their LEMS extensions (**Fig. 5**), allowing NeuroML to remain simulator-independent while supporting multiple simulation back-ends.

Newly created elements that may be of interest to the wider research community can be submitted to the NeuroML Editorial Board for inclusion into the standard. The standard, therefore, evolves as new model elements are added, and improved versions of the standard and associated software tool chain are regularly released to the community.

## NeuroML is maintained by an international, collaborative community

NeuroML is an open community standard, maintained collectively by a diverse set of stakeholders. This ensures that NeuroML supports the myriad of use cases generated by a multi-disciplinary computational modeling community.

The NeuroML Scientific Committee [14] and the elected NeuroML Editorial Board [15] oversee the standard, the core tools, and the initiative. The Scientific Committee sets the scientific focus of the NeuroML initiative. It ensures that the standard represents the state of the art—that it can encapsulate the latest knowledge in neuronal anatomy and physiology in their corresponding model components. The Scientific Committee also defines the governance structure of the initiative, and works with the wider scientific community to gather feedback on NeuroML and promote its use. The Editorial Board manages the day to day development and maintenance of LEMS, the NeuroML schema, the core software tools, and critical resources such as the documentation. The Editorial Board works with simulator developers in the extended ecosystem to help make tools NeuroML compliant by testing reference implementations and answering technical queries about NeuroML and the core software tools.

NeuroML is strongly linked to global standardization initiatives. NeuroML is an endorsed INCF (***Abrams et al., 2022***) community standard (***Martone et al., 2019***) and is one of the core standards of the international COMBINE initiative (***Hucka et al., 2015***), which supports the development of other standards in computational biology as well (e.g. SBML (***Hucka et al., 2003***) and CellML (***Lloyd et al., 2004***)). Participation in these organizations guarantees that NeuroML follows current best practices in standardization, and remains linked to and interoperable with other standards wherever possible.

Outreach and training is critical to the uptake and improvement of NeuroML. This includes in person (and in recent years online) tutorials, development workshops, and presentations at the annual meetings of multiple organizations such as, INCF, COMBINE, and Organization for Computational Neuroscience (OCNS). Under the umbrella of the INCF, the NeuroML community also participates in the Google Summer of Code initiative where candidates have converted published computational models into the standardized NeuroML format. These standardized models are made available to the community on the OSB platform and promoted at OSB workshops.

Communication within the NeuroML community is facilitated via a public mailing list[16] for asynchronous communication and announcements while open chat channels on Gitter (now Matrix/Element[17]) enable users and developers more immediate access to the NeuroML team for troubleshooting and general discussion. Finally, all software repositories hosted on GitHub also have issue trackers which are used for software specific queries. These channels and the aforementioned training activities provide the opportunity to learn more about NeuroML, help with standardizing models, and making tools NeuroML compliant. A community Code of Conduct[18] sets the standards of communication and behavior expected on all community channels.

A core aim of NeuroML is to enable Open Science and ensure models in computational neuroscience are FAIR. To this end, all development and discussions related to NeuroML are done publicly. The schema, all core software tools, and related resources such as documentation are made freely available under suitable Free/Open Source Software (FOSS) licenses on public platforms. Everyone can, therefore, use, modify, study, and share all NeuroML artifacts without restriction. Users and developers are encouraged to contribute modifications and improvements to the schema and core tools and to participate in the general maintenance and release process.

## Discussion

The model description language NeuroMLv2 has matured into a widely adopted community standard for computational neuroscience. Its modular, hierarchical structure can define a wide range of neuronal and circuit model types including simplified representations and those with a high degree of biological detail. The standardized, machine readable format of the NeuroMLv2/LEMS framework provides a flexible, common language for communicating between a wide range of tools and simulators used to create, validate, visualize, analyze, simulate, share and reuse models. By enabling this interoperability, NeuroMLv2 has spawned a large ecosystem of interacting tools that cover all stages of the model development life cycle, bringing greater coherence to a previously fragmented landscape. Moreover, the modular nature of the model components and hierarchical structure conferred by NeuroMLv2, combined with the flexibility of coding in Python, has created a powerful “building block” approach for constructing standardized models from scratch. This can be facilitated by harnessing the NeuroMLv2 tool ecosystem and the multi-level automated validation of NeuroMLv2/LEMS based models. NeuroML has therefore evolved from a standardized archiving format into a mature language that supports an ecosystem of tools for the creation and execution of models that support the FAIR principles and promote open, transparent and reproducible science.

### Evolution of NeuroML and emergence of the NeuroMLv2 tool ecosystem

NeuroML was first conceived (***Goddard et al., 2001***) and developed (***Gleeson et al., 2010***) as a declarative XML based framework for defining biophysical models of neurons and networks in a standardized form in order to compare model properties across simulators and to promote transparency and reuse. NeuroML version 1 achieved these aims and was mainly used to archive and visualize existing models (***Gleeson et al., 2010***). Building on this, the subsequent development of the NeuroMLv2/LEMS framework provided a way to describe models as a hierarchical set of components with dimensional parameters and state variables, so that their structure and dynamics are fully machine readable (***Cannon et al., 2014***). This enabled models to be losslessly mapped to other representations, greatly facilitating interoperability between tools through read-write and automated code generation (***Blundell et al., 2018***). As NeuroMLv2 matured and became a community standard recognized by the INCF with a formal governance structure, an increasingly wide range of models and modeling tools have been developed or modified to be NeuroMLv2 compliant (Appendix 1 **Tables 1**, 2 and 6). The core tools, maintained directly by the NeuroML developers (**Fig. 4**), provide functionality to read, modify, or create new NeuroML models, as well as to analyze and visualize, and simulate the models. Moreover, there are now a larger number of tools that have been developed by other members of the

community (**Fig. 3**), including a neuronal simulator designed specifically for NeuroMLv2 (***Panagiotou et al., 2022***). The emergence of an ecosystem of NeuroMLv2 compliant tools enables modelers to build tool chains that span the model life cycle and build and reuse standardized models.

## NeuroML and other standards in computational neuroscience

A number of other standards and formats exist to support computational modeling of neuronal systems. Whereas NeuroML is a modular, declarative simulator independent standard for biophysical neuronal modelling, PyNN (***Davison et al., 2009***) and SONATA (***Dai et al., 2020***) provide a procedural Python based simulator independent API and a framework for efficiently handling large scale network simulations, respectively. Even though there is some overlap in the functionality provided by these standards, they each target distinct use cases and have their own goals and features. The teams developing these standards work in concert to ensure that they remain interoperable with each other, frequently sharing methods and techniques (***Dai et al., 2020***). This allows researchers to use their standard of choice and easily combine with another if the need arises. PyNN and SONATA are therefore integral parts of the wider NeuroML ecosystem.

## Why using NeuroML and Python facilitates the construction of FAIR models

The modular and hierarchical structure of NeuroMLv2, when combined with Python, provides a powerful combination of structured declarative elements and flexible procedural approaches that enables a "Lego-like" building block approach for constructing biologically detailed models (***Cayco-Gajic et al., 2017***; ***Billings et al., 2014***; ***Kriener et al., 2022***; ***Gurnani and Silver, 2021***). This has been facilitated by the development of pyNeuroML, which provides a single installable package which offers direct access to a range of functionality for handling NeuroML models (Box 1). Moreover, the web-based documentation of NeuroMLv2, with multiple Python scripts illustrating the usage of the language and associated tools (**Table 1**), has recently been updated and expanded (*https://docs.neuroml.org*). This provides a core resource for both new and experienced users of NeuroML facilitating its use in model building. As the examples of this resource illustrate, building models using NeuroMLv2 is efficient and intuitive, as the model components are pre-made and how they fit together specified. The structured format allows APIs like libNeuroML to incorporate features such as auto-completion and inline validation of model parameters and structure as scripts are being developed. In addition, automated multi-stage model validation ensures the code, equations and internal structure are validated against the NeuroML schema minimizing human errors and model simulations outputs are within acceptable bounds (**Fig. 6**). The NeuroMLv2 ecosystem also provides convenient ways to visualize and inspect the inner structure of models. pyNeuroML provides Python functions and corresponding command line utilities to view neuronal morphology (**Fig. 7**), neuronal electrophysiology (**Fig. 9**), circuit connectivity and schematics (**Fig. 8**). In addition, custom analysis pipelines and advanced neuroinformatics resources can easily be built using the APIs. For example, loading a NeuroML model of a neuron into OSB enables visualization of the morphology and the spatial distribution of ionic conductance over the membrane as well as inspection of the conductance state variables, while the connectivity and synaptic weight matrices can be automatically displayed for circuit models (**Fig. 7**, ***Gleeson et al***. (***2019***)). Such features of OSB, which are made possible by the structured format of NeuroMLv2, promote model transparency, reproducibility and sharing. By enabling the development and sharing of well tested and transparent models the wider NeuroMLv2 ecosystem promotes Open Science.

## Limitations of NeuroML and current work

A limitation of any standardized framework is that there will always be models and model elements that fall outside the current scope of the standard. While NeuroML suffers from this limitation, the underlying LEMS based framework provides a flexible route to develop a wide

range of new types of physio-chemical model (***Cannon et al., 2014***). This is relatively straightforward if the new model component, such as a synaptic plasticity mechanism, fits within the existing hierarchical structure of NeuroMLv2, as the new type of synaptic element can build on an existing base synapse type, which specifies the relevant input and outputs (e.g. local voltage and synaptic current). For more radical shifts in model types (e.g. neuronal morphologies that grow during learning), that do not fit neatly into the current NeuroMLv2 schema, structural changes to the language would be required. This route is more involved as the pros and cons of changes to the structure of NeuroMLv2 would need to be considered by the Scientific Committee and, if approved, implemented by the Editorial Board.

While the current scope of NeuroMLv2 encompasses models of spiking neurons and networks at different levels of biological detail, plans are in place to extend its scope to include more abstract, rate-based models of neuronal populations (e.g. see ***Wilson and Cowan*** (***1972***); ***Mejias et al***. (***2016***) in Appendix 1 Table 6). Additionally, work is under way to extend current support for SBML (***Hucka et al., 2003***) based descriptions of chemical signaling pathways (***Cannon et al., 2014***), to enable more complete biochemical descriptions of sub-cellular activity in neurons and synapses.

There is a growing interest in the field for the efficient generation and serialization of large scale network models, containing numbers of neurons closer to their biological equivalents (***Markram et al., 2015***; ***Billeh et al., 2020***; ***Einevoll et al., 2019***). While a number of applications in the NeuroML ecosystem support large scale model generation (e.g. NetPyNE, neuroConstruct, PyNN), the default serialization of NeuroML (XML) is inefficient for reading/writing/storing such extensive descriptions. NeuroML does have an internal format for serializing in the binary format HDF5 (see Methods), but has also recently added support for export of models to the SONATA data format (***Dai et al., 2020***), allowing efficient serialization of large scale models. While individual instances of large scale models are useful, the ability to generate families of these for multiple simulation runs, and more particularly a way to encapsulate, examine and reuse “recipes” for network models, is also required. A prototype package, NeuroMLlite[19], has been developed which allows these concise network templates to be described and multiple instances of networks to be generated, and facilitates interaction with simulation platforms and efficient serialization formats.

As discoveries and insights in neuroscience inform machine learning and visa versa, there is an increasing need to develop a common framework for describing both biological and artificial neural networks. Model Description Format (MDF) has been developed to address this (***Gleeson et al., 2023***). This initiative has developed a standardized format, along with a Python API, which allows the specification of artificial neural networks (e.g. Convolutional Neural Networks, Recurrent Neural Networks) and biological neurons using the same underlying entities. Support for mapping MDF to/from NeuroMLv2/LEMS has been included from the start. This work will enable deeper integration of computational neuroscience and “brain-inspired” networks in Artificial Intelligence (AI).

## Conclusion and vision for the future

NeuroMLv2 is already a mature community standard that provides a framework for standardizing biologically detailed neuronal network models. By providing a stable, common framework defining the core entities required for biologically detailed neuronal modeling, NeuroML has spawned an ecosystem of tools that span all stages of the model development life cycle. In the short term, we envision the functionality of NeuroML to expand further and for new online resources that facilitate the construction of FAIR models using pyNeuroML to be taken up by the community. The NeuroML development team are also beginning to explore how to combine NeuroML-based circuit models with musculo-skeletal simulations to enable models of the neural control of behavior. In the longer term, developing seamless interfaces between NeuroML and

other domain specific standards will enable the development of more holistic models of the neural control of body systems across a wide range of organisms, as well as greater exchange of models and insights between computational neuroscience and AI.

## Methods

NeuroMLv2 is formally specified by the NeuroMLv2 XML schema, which defines the allowed structure of XML files which comply to the standard, and the LEMS ComponentType definitions, which define the internal state variables of the underlying elements, providing a machine-readable specification of the time evolution of model components. This core specification is backed up by a suite of software tools that support the model life cycle, and the accompanying usage and development documentation.

We illustrate the key parts of this framework using the HindmarshRose cell model (***Hindmarsh et al***. (***1984***), **Fig. 10**), which, as an abstract point neuron model, serves as an appropriate simple NeuroMLv2 ComponentType.

### The NeuroML XML Schema

We begin with the NeuroMLv2 standard. The standard consists of two parts, each serving different functions:

1. the NeuroMLv2 XML schema
2. corresponding LEMS component type definitions

The NeuroMLv2 schema is a language independent data model that constrains the structure of a NeuroMLv2 *model description*. The NeuroML schema is formally described as an XML Schema document[20] in the XML Schema Definition (XSD) format, a recommendation of the World Wide Web Consortium (W3C)[21]. An XML document that claims to conform to a particular schema can be *validated* against the schema. All NeuroMLv2 model descriptions can therefore, be validated against the NeuroMLv2 schema.

The basic building blocks of an XSD schema are “simple” or “complex” types and their “attributes”. All types are created as “extensions” or “restrictions” of other types. Complex types may contain other types and attributes whereas simple types may not. **Fig. 11** shows some example types defined in the NeuroMLv2 schema. For example, the Nml2Quantity_none simple type restricts the in-built “string” type using a regular expression “pattern” that limits what string values it can contain. The type is Nml2Quantity_none is to be used for unit-less quantities (e.g. 3, 6.7, -1.1e-5) and the restriction pattern for translates to “a string that may start with a hyphen (negative sign), followed by any number of numerical characters (potentially containing a decimal point) and a string containing capital or small ‘e’ (to specify the exponent)”. The restriction pattern for the Nml2Quantity_voltage type is similar, but must be followed by a “V” or “mV”. In this way, the restriction ensures that a value of type “Nml2Quantity_voltage” represents a physical voltage quantity with units “V” (volt) or “mV” (millivolt). Furthermore, a NeuroMLv2 model description that uses a voltage value that does not match this pattern, for example “0.5 *s*”, will be invalid.

The example of a complex type in **Fig. 11** is the HindmarshRose1984Cell type that extends the BaseCellMembPotCap complex type (the base type for any cell producing a membrane potential v with a capacitance parameter C), and defines a number of new “required” (compulsory) attributes. These attributes are of simple types—these are all unit-less quantities apart from v_scaling, which has dimension voltage. Note that inherited attributes are not re-listed in the complex type definition—the compulsory capacitance attribute, C, is inherited here from BaseCellMembPotCap.

**a.** NeuroML model description serialization

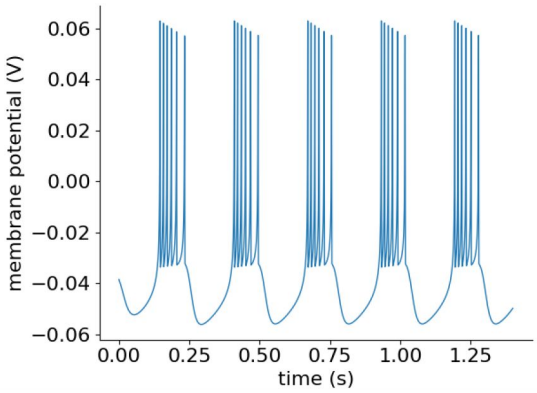
```
<neuroml xmlns="http://www.neuroml.org/schema/neuroml2"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.neuroml.org/schema/neuroml2
    https://raw.github.com/NeuroML/NeuroML2/development/Schemas/NeuroML2/NeuroML_v2.3.xsd"
    id="HindmarshRoseNeuron">

    <hindmarshRose1984Cell id="hr_regular_bursting" C="28.57142857pF" a="1.0" b="3.0" c="-3.0"
    d="5.0" s="4.0" x1="-1.3" r="0.002" x0="-1.1" y0="-9" z0="1.0" v_scaling="1.0mV"/>

    <pulseGenerator id="pulseGen_0" delay="0s" duration="1000s" amplitude="5nA"/>

    <network id="HRNet">
        <population id="HRPop0" component="hr_regular_bursting" size="1"/>
        <explicitInput target="HRPop0[0]" input="pulseGen_0" destination="synapses"/>
    </network>
</neuroml>
```

**b.**




## Figure 10.

Example model description of a HindmarshRose1984Cell NeuroML component. **(a)** XML serialization of the model description containing the main hindmarshRose1984Cell element, with a set of parameters which result in regular bursting. A current clamp stimulus is applied using a pulseGenerator, and a population of one cell is added with this in a network. This XML can be validated against the NeuroML Schema. **(b)** Membrane potentials generated from a simulation of the model in (a). The LEMS simulation file to execute this is shown in **Fig. 14**.

```
<xs:simpleType name="Nml2Quantity_none">  <!-- For dimensionless parameters -->
  <xs:restriction base="xs:string">
    <xs:pattern value="-?([0-9]*(\.[0-9]+)?)([eE]-?[0-9]+)?"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="Nml2Quantity_voltage">  <!-- For params with dimension voltage -->
  <xs:restriction base="xs:string">
    <xs:pattern value="-?([0-9]*(\.[0-9]+)?)([eE]-?[0-9]+)?[\s]*(V|mV)"/>
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="HindmarshRose1984Cell">
  <xs:annotation>
    <xs:documentation>The Hindmarsh Rose model is a simplified point cell model which
        captures complex firing patterns of single neurons, such as
        periodic and chaotic bursting...
    </xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="BaseCellMembPotCap">
      <xs:attribute name="a" type="Nml2Quantity_none" use="required"/>
      <xs:attribute name="b" type="Nml2Quantity_none" use="required"/>
      <xs:attribute name="c" type="Nml2Quantity_none" use="required"/>
      <xs:attribute name="d" type="Nml2Quantity_none" use="required"/>
      <xs:attribute name="s" type="Nml2Quantity_none" use="required"/>
      <xs:attribute name="x1" type="Nml2Quantity_none" use="required"/>
      <xs:attribute name="r" type="Nml2Quantity_none" use="required"/>
      <xs:attribute name="x0" type="Nml2Quantity_none" use="required"/>
      <xs:attribute name="y0" type="Nml2Quantity_none" use="required"/>
      <xs:attribute name="z0" type="Nml2Quantity_none" use="required"/>
      <xs:attribute name="v_scaling" type="Nml2Quantity_voltage" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

## Figure 11.

Type definitions taken from the **NeuroMLv2 schema**, which describes the structure of NeuroMLv2 elements. **Top:** "simple" types may not include other elements or attributes. Here, the Nml2Quantity_none and Nml2Quantity_voltage types define restrictions on the default string type to limit what strings can be used as valid values for attributes of these types. **Bottom:** example of a "complex" type, the HindmarshRose cell model (***Hindmarsh et al., 1984***), that can also include other elements of other types, and extend other types.

The NeuroMLv2 schema serves a number of critical functions. A variety of tools and libraries support the validation of files against XSD schema definitions. Therefore, the NeuroMLv2 schema enables the validation of model descriptions *prior to simulation* (**Fig. 6**). XSD schema definitions, as language independent data models, also allow the generation of APIs in different languages. More information on how APIs in different languages are generated using the NeuroMLv2 XSD schema definition is provided in later sections.

The NeuroMLv2 XSD schema is also released and maintained as a versioned artifact, similar to the software packages. The current version is 2.3, and can be found in the NeuroML2 repository on GitHub[22].

## LEMS ComponentType definitions

The second part of the NeuroMLv2 standard consists of the corresponding LEMS ComponentType definitions. Whereas the XSD Schema describes the *structure* of a NeuroMLv2 model description, the LEMS ComponentType definitions formally describe the *dynamics* of the model elements.

LEMS (***Cannon et al., 2014***) is a domain independent general purpose machine-readable language for describing models and their simulations. A complete description of LEMS is provided in ***Cannon et al***. (***2014***) and in our documentation[23]. Here, we limit ourselves to a short summary necessary for understanding the NeuroMLv2 ComponentType definitions.

LEMS allows the definition of new model types referred to as ComponentTypes. These are formal descriptions of how a generic model element of that type behaves (the "dynamics"), *independent of the specific set of parameters in any instance*. To describe the dynamics, such descriptions must list any necessary parameters that are required, as well as the time-varying state variables. The dimensions of these parameters and state variables must be specified, and any expressions involving them must be dimensionally consistent. An instance of such a generic model is termed a Component, and can be instantiated from a ComponentType by providing the necessary parameters. One can can think of ComponentTypes as user defined data types similar to "classes" in many programming languages, and Components as "objects" of these types with particular sets of parameters. Types in LEMS can also extend other types, enabling the construction of a hierarchical library of types. In addition, since LEMS is designed for model simulation, ComponentType definitions also include other simulation related features such as Exposures, specifying quantities that may be accessed/recorded by users.

There is a one-to-one mapping between elements specified in the NeuroML XSD schema, and LEMS ComponentTypes, with the same parameters specified in each. The addition of new model elements to the NeuroML standard, therefore, requires the addition of new type definitions to both the XSD schema and the LEMS definitions. New user defined ComponentTypes, however, can be easily defined in LEMS and used freely in models, and these do not need to be added to the standard before use. The only limitation here is that new user defined ComponentTypes cannot be validated against the NeuroML schema, since their type definitions will not be included there.

**Fig. 12** shows the ComponentType definition for the HindmarshRose1984Cell model element. Here, the HindmarshRose1984Cell ComponentType extends baseCellMembPotCap and inherits its elements. The ComponentType includes a number of parameters that users must provide when creating a new instance (component): *a, b, c, d, r, s, x*1, *v_scaling*.

Other parameters, *x*0, *y*0, and *z*0 are used to initialize the three state variables of the model, *x, y, z*. *x* is the proxy for the membrane potential of the cell used in the original formulation of the model (***Hindmarsh et al., 1984***) and is here scaled by a factor *v_scaled* to expose a more physiological value for the membrane potential of the cell in StateVariable *v*. A Constant MSEC is defined to hold the value of 1 *ms* for use in the ComponentType. Next, an Attachment is used to enable the

```
<ComponentType name="hindmarshRose1984Cell" extends="baseCellMembPotCap" description="The Hindmarsh Rose
  model">
  <Parameter name="a" dimension="none" description="cubic term in x nullcline"/>
  <Parameter name="b" dimension="none" description="quadratic term in x nullcline"/>
  <Parameter name="c" dimension="none" description="constant term in y nullcline"/>
  <Parameter name="d" dimension="none" description="quadratic term in y nullcline"/>
  <Parameter name="r" dimension="none" description="timescale separation between slow and fast subsystem (r
  greater than 0; r much less than 1)"/>
  <Parameter name="s" dimension="none" description="related to adaptation"/>
  <Parameter name="x1" dimension="none" description="related to the system's resting potential"/>
  <Parameter name="v_scaling" dimension="voltage" description="scaling of x for physiological membrane
  potential"/>

  <!-- Initial Conditions -->
  <Parameter name="x0" dimension="none"/>
  <Parameter name="y0" dimension="none"/>
  <Parameter name="z0" dimension="none"/>

  <Constant name="MSEC" dimension="time" value="1ms"/>

  <Attachments name="synapses" type="basePointCurrent"/>

  <Exposure name="x" dimension="none"/>
  <Exposure name="y" dimension="none"/>
  <Exposure name="z" dimension="none"/>
  <Exposure name="phi" dimension="none"/>
  <Exposure name="chi" dimension="none"/>
  <Exposure name="rho" dimension="none"/>
  <Exposure name="spiking" dimension="none"/>
  <Dynamics>
    <StateVariable name="v" dimension="voltage" exposure="v"/>
    <StateVariable name="y" dimension="none" exposure="y"/>
    <StateVariable name="z" dimension="none" exposure="z"/>
    <StateVariable name="spiking" dimension="none" exposure="spiking"/>

    <DerivedVariable name="iSyn" dimension="current" exposure="iSyn" select="synapses[*]/i" reduce="add" />
    <DerivedVariable name="x" dimension="none" exposure="x" value="v / v_scaling"/>
    <DerivedVariable name="phi" dimension="none" exposure="phi" value="y - a * x^3 + b * x^2"/>
    <DerivedVariable name="chi" dimension="none" exposure="chi" value="c - d * x^2 - y"/>
    <DerivedVariable name="rho" dimension="none" exposure="rho" value="s * ( x - x1 ) - z"/>
    <DerivedVariable name="iMemb" dimension="current" exposure="iMemb"
                     value="(C * (v_scaling * (phi - z) / MSEC)) + iSyn"/>

    <TimeDerivative variable="v" value="iMemb/C"/>
    <TimeDerivative variable="y" value="chi / MSEC"/>
    <TimeDerivative variable="z" value="r * rho / MSEC"/>

    <OnStart>
      <StateAssignment variable="v" value="x0 * v_scaling"/>
      <StateAssignment variable="y" value="y0"/>
      <StateAssignment variable="z" value="z0"/>
    </OnStart>
    <OnCondition test="v .gt. 0 .and. spiking .lt. 0.5">
      <StateAssignment variable="spiking" value="1"/>
      <EventOut port="spike"/>
    </OnCondition>
    <OnCondition test="v .lt. 0">
      <StateAssignment variable="spiking" value="0"/>
    </OnCondition>
  </Dynamics>
</ComponentType>
```

**Figure 12.**

**LEMS ComponentType definition**

of the HindmarshRose cell model (***Hindmarsh et al., 1984***).

addition of entities that would provide external inputs to the ComponentType. Here, synapses are Attachments of the type basePointCurrent and provide synaptic current input to this ComponentType.

The Dynamics block lists the mathematical formalism required to simulate the ComponentType. By default, variables defined in the Dynamics block are private, i.e., they are not visible outside the ComponentType. To make these visible to other ComponentTypes and to allow users to record them, they must be connected to Exposures. Exposures for this ComponentType include the three state variables, but also the internal derived variables, which while not used by other components, are useful in inspecting the ComponentType and its dynamics. An extra exposure, *spiking*, is added to allow other NeuroML components access to the spiking state of the cell that will be determined in the Dynamics block.

A number of StateVariable definitions are followed by DerivedVariables, variables whose values depend on other variables. The total synaptic current, *iSyn*, is a summation of all the synaptic currents, *i* received by the synapses that may be attached on to this ComponentType. The add value of the reduce field tells LEMS that when there are multiple values, they should all be summed. As noted, *x* is a scaled version of the membrane potential variable, *v*. This is followed by the three derived variables, *ph i, ch i, rho* where:

$$phi = y - ax^3 + bx^2 \tag{1}$$

$$chi = c - dx^2 - y \tag{2}$$

$$rho = s(x - x1) - z \tag{3}$$

The total membrane potential of the cell, *iMemb* is calculated as the sum of the capacitive current and the synaptic current:

$$iMemb = \frac{C(v\_scaling(phi - z))}{\text{MSEC}} + iSyn \tag{4}$$

*v, y, z* are TimeDerivatives, with the “value” representing the rate of change of each variable:

$$dv/dt = iMemb/C \tag{5}$$

$$dy/dt = chi/\text{MSEC} \tag{6}$$

$$dz/dt = (r \times rho)/\text{MSEC} \tag{7}$$

The final few blocks set the initial state of the component (OnStart),

$$v = x0 \times v\_scaling \tag{8}$$

$$y = y0 \tag{9}$$

$$z = z0 \tag{10}$$

and define conditional expressions to set the spiking state of the cell:

$$spiking = \begin{cases} 1 & \text{if } (v > 0) \wedge (spiking < 0.5) \\ 0 & \text{if } (v < 0) \end{cases} \tag{11}$$

Both the XSD schema and the LEMS ComponentType definitions enable model validation. However, despite some overlap, they support different types of validation. Whereas the XSD schema allows for the validation of *model descriptions* (e.g. the XML files), the LEMS ComponentType definitions enable validation of *model instances*, i.e., the "runnable" instances of models that are constructed once components have been created by instantiating ComponentTypes with the necessary parameters, and various attachments created between source and target components. A model description may be used to create a number of different model instances for simulation. Indeed, it is common practice to run models that include stochasticity with different seeds for random number generators to verify the robustness of simulation results. Thus, the validation of dimensions and units that LEMS carries out is done only after a runnable instance of a model has been created.

The LEMS ComponentType definitions for NeuroMLv2 are also maintained as versioned files that are updated along with the XSD schema. These can also be seen in the NeuroMLv2 GitHub repository[24]. An index of the ComponentTypes included in version 2.3 of the NeuroML standard, with links to online documentation, is also provided in Appendix 1 Tables 4 and 5.

## NeuroML APIs

The NeuroMLv2 software stack relies on the NeuroML APIs that provide functionality to read, write, validate, and inspect NeuroML models. The APIs are programmatically generated from the machine readable XSD schema, thus ensuring that the class for defining a specific NeuroML element in a given language (e.g. Java) has the correct set of fields with the appropriate type (e.g. float or string) corresponding to the allowed parameters in the corresponding NeuroML element. NeuroMLv2 currently provides APIs in a number of languages—Python (libNeuroML which is generated via generateDS[25]), Java (org.neuroml.model via JAXB XJC[26]), C++ (NeuroML_CPP via XSD[27]) and MATLAB (NeuroML-Toolbox which accesses the Java API from MATLAB), and APIs for other languages can also be easily generated as required. LEMS is also supported by a similar set of APIs—PyLEMS in Python, and jLEMS in Java—and since a NeuroMLv2 model description is a set of LEMS Components, the LEMS APIs also support them (e.g. the hindmarshRose1984Cell example in **Fig. 10** could be loaded by jLEMS and treated as a LEMS Component).

**Fig. 13** shows the use of the NeuroML Python API to describe a model with one HindmarshRose cell. In Python, the instances of ComponentTypes, their Components, are represented as Python objects. The hr0 Python variable stores the created HindmarshRose1984Cell component/object. This is added to a Population, pop0 in the Network net. The network also includes a PulseGenerator with amplitude 5 nA as an ExplicitInput that is targeted at the cell in the population. The model description is serialized to XML (**Fig. 10**) and validated. Note that, as the standard convention for classes in Python is to use capitalized names, HindmarshRose1984Cell is used in Python, which is serialized as <hindmarshRose1984Cell> in the XML. Users can either share the Python script itself, or the XML serialization. Any valid XML serialization can be also loaded into a Python object model and modified.

XML is the default serialization of NeuroML, and all existing APIs can read and write the format (and it should be seen as a minimal requirement for new APIs to support XML). There is however an alternative HDF5[28] based serialization of NeuroML files which is supported by both libNeuroML and the Java API, org.neuroml.model[29]. This format is based on an efficient representation of cell positions and connectivity data as HDF5 data sets, which can be serialized in compact binary format, and loaded into memory for optimized access (e.g. as numpy arrays in libNeuroML). This reduces the size of the saved files for large scale networks, and speeds up loading/writing models, eliminating the need to parse/generate large text files containing XML. Models serialized in this format can be loaded and transformed to simulator code in the same way as XML based models by the Java and Python APIs.

```
nml_doc = component_factory("NeuroMLDocument", id="HindmarshRoseNeuron")
hr0 = nml_doc.add("HindmarshRose1984Cell", id="hr_regular", a="1.0", b="3.0", c="-3.0", d="5.0",
    s="4.0", x1="-1.3", r="0.002", x0="-1.1", y0="-9", z0="1.0", C="28.57142857pF",
    v_scaling="35.0mV")
net = nml_doc.add("Network", id="HRNet", validate=False)
```

Create a population of cells (1 cell)

```
pop0 = net.add("Population", id="HRPop0", component=hr0.id, size=1)
```

Add external stimuli to the population

```
pg = nml_doc.add("PulseGenerator", id="pulseGen_%i" % 0, delay="0s", duration="1000s",
    amplitude="5nA")
exp_input = net.add("ExplicitInput", target="%s[%i]" % (pop0.id, 0), input=pg.id,
    destination="synapses")
```

Save (serialize) the model to a file

```
nml_file = 'hindmarshrose1984_single_cell_network.nml'
writers.NeuroMLWriter.write(nml_doc, nml_file)
```

Validate the model

```
validate_neuroml2(nml_file)
```

**Figure 13.**

Example model description of a HindmarshRose1984Cell NeuroML component in Python, using parameters for regular bursting. This script generates the XML in **Fig. 10**.

## Simulating NeuroML models

The model description shown in **Fig. 10** contains no information about how it is to be simulated, or on the dynamics of each model component. Providing this simulation information and linking in the ComponentType definition requires creating a LEMS file to fully specify the simulation. **Fig. 14** shows the use of utilities included in the Python pyNeuroML package to describe a LEMS simulation of the HindmarshRose model defined in **Fig. 10**. The LEMSSimulation object includes simulation specific information, such as the duration of the simulation, the integration time step, and the seed value. It also allows the specification of files for the storage of data recorded from the simulation. In this example, we record the membrane potential, *v*, of our cell in its population, HRPop0[0]. Similar to the NeuroMLv2 model description, the simulation object can also be serialized to XML for storage and sharing (**Fig. 14**, bottom).

As noted previously, NeuroML/LEMS model and simulation descriptions are machine readable and simulator independent. They can, therefore, be translated into specific formats for simulation on different simulation backends. The core tool for simulating NeuroML/LEMS models is jNeuroML, which is also made available to Python users via pyNeuroML. In addition to functions for programmatic access, jNeuroML and pyNeuroML also provide the jnml and pynml command line tools (Box 1).

jNeuroML supports all five simulator engine categories (**Fig. 5**). It includes the jLEMS reference LEMS simulator that can simulate LEMS models that do not use the cable equation. It can therefore, directly simulate simpler models. For example, the HindmarshRose model shown in **Figs. 13** and **14** is simulated directly in jNeuroML. jNeuroML can also pass simulations to the EDEN simulator (***Panagiotou et al., 2022***) for direct simulation. For simulators that require the conversion of NeuroML/LEMS simulations into either scripts which allow importing by the simulator's own methods (e.g. NetPyNE (***Dura-Bernal et al., 2019***)) or full conversion to native simulator scripts (e.g. NEURON (***Hines and Carnevale, 1997***)), jNeuroML includes the org.neuroml.export library[30]. This library translates the complete, expanded LEMS model simulation instance that is available to it in jNeuroML (via jLEMS) and translates it into the required format. Thus, supporting a new simulation back end that requires translation of NeuroML/LEMS into another format requires the addition of new "writers" to the org.neuroml.export library. jNeuroML also includes the org.neuroml.import[31] library that converts from other formats (e.g. SBML (***Hucka et al., 2003***)) to LEMS for combination with NeuroML models.

All NeuroML and LEMS software packages are made available under FOSS licenses. The source code for all NeuroML packages and the standard can be obtained from the NeuroML GitHub organization[32]. The NeuroML Python API[33] was developed in collaboration with the NeuralEnsemble initiative[34], which also maintains other commonly used Python packages such as PyNN (***Davison et al., 2009***), Neo (***Garcia et al., 2014***) and Elephant (***Denker et al., 2018***). LEMS packages are available from the LEMS GitHub organization[35].

To ensure replication and reproduction of studies, it is important to note the exact versions of software used in studies. For NeuroML and LEMS packages, archives of each release along with citations are published on Zenodo[36] to enable researchers to cite them in their work.

## Documentation

A standard and its accompanying software ecosystem must be supported by comprehensive documentation if it is to be of use to the research community. The primary NeuroML documentation for users that accompanies this paper has been consolidated into a JupyterBook (***Executable Books Community, 2020***) at *https://docs.neuroml.org*. This includes explanations of NeuroML and computational modeling concepts, interactive tutorials with varying levels of

Create a simulation of the model

```
simulation_id = "example-single-hindmarshrose1984cell-sim"
simulation = LEMSSimulation(sim_id=simulation_id, duration=1.4e3, dt=0.0025,
    simulation_seed=123)
simulation.assign_simulation_target(net.id)
simulation.include_neuroml2_file(nml_file)
```

Record membrane potential to an output file

```
simulation.create_output_file("output0", "%s.v.dat" % simulation_id)
simulation.add_column_to_output_file("output0", 'HRPop0[0]', 'HRPop0[0]/v')
```

Save the simulation to file and run it in jNeuroML/jLEMS

```
lems_simulation_file = simulation.save_to_file()
pynml.run_lems_with_jneuroml(lems_simulation_file, max_memory="2G", nogui=True, plot=False)
```

LEMS simulation description serialization

```
<Lems>
    <!-- Specify which component to run -->
    <Target component="example-single-hindmarshrose1984cell-sim"/>

    <!-- Include core NeuroML2 ComponentType definitions -->
    <Include file="Cells.xml"/>
    <Include file="Networks.xml"/>
    <Include file="Simulation.xml"/>

    <Include file="hindmarshrose1984_single_cell_network.nml"/>

    <Simulation id="example-single-hindmarshrose1984cell-sim" length="1400.0ms" step="0.0025ms"
    target="HRNet" seed="123">  <!-- Note seed: ensures same random numbers used every run -->
        <OutputFile id="output0" fileName="example-single-hindmarshrose1984cell-sim.v.dat">
            <OutputColumn id="HRPop0[0]" quantity="HRPop0[0]/v"/>
        </OutputFile>
    </Simulation>
</Lems>
```

**Figure 14.**

An example simulation of the HindmarshRose model description shown in **Fig. 13**, with the LEMS serialization shown at the bottom.

complexity, information about tools and what functions they provide to support different stages of the model life cycle. The JupyterBook framework supports "executable" documentation through the inclusion of interactive Jupyter notebooks which may be run in the users' web browser on free services such as OSBv2, *Binder.org*[37] (***Project Jupyter et al., 2018***) and Google Colab[38]. Finally, the machine readable nature of the schema and LEMS also enables the automated generation of human readable documentation for the standard and low level APIs (**Fig. 15**) along with their examples[39]. In addition, the individual NeuroML software packages each have their own individual documentations (e.g. pyNeuroML[40], libNeuroML[41]).

As with the rest of the NeuroML ecosystem, the documentation is hosted on GitHub[42], licensed under a FOSS license, and community contributions to it are welcomed. A PDF version of the documentation can also be downloaded for offline use[43].

## Acknowledgements

We thank all the members of the NeuroML Community who have contributed to the development of the standard over the years, have added support for the language to their applications, or who have converted published models to NeuroML. We would particularly like to thank the following for contributions to the NeuroML Scientific Committee: Upi Bhalla, Avrama Blackwell, Hugo Cornells, Robert McDougal, Lyle Graham, Cengiz Gunay and Michael Hines. The following have also contributed to developments related to the named tools/simulators/resources: EDEN - Mario Negrello and Christos Strydis, SONATA - Anton Arkhipov and Kael Dai, MOOSE - Subhasis Ray, NeuroML-DB - Justas Birgiolas, *NeuroMorpho.Org* - Giorgio Ascoli, N2A - Fred Rothganger, pyLEMS - Gautham Ganapathy, MDF - Manifest Chakalov, libNeuroML and NeuroTune - Mike Vella, Open Source Brain - Matt Earnshaw, Adrian Quintana and Eugenio Piasini, SciUnit/NeuronUnit - Richard C Gerkin, Brian - Marcel Stimberg and Dominik Krzemiński, Arbor - Nora Abi Akar, Thorsten Hater and Brent Huisman, BluePyOpt - Jaquier Aurélien Tristan and Werner van Geit, C++/MATLAB APIs - Jonathan Cooper. We thank Rokas Stanislavos, András Ecker, Jessica Dafflon, Ronaldo Nunes, Anuja Negi and Shayan Shafquat for their work converting models to NeuroML format as part of the Google Summer of Code program. We also thank Diccon Coyle for feedback on the manuscript.

## Funding


| Funder | Grant reference number | Author |
|---|---|---|
| Wellcome | 212941 | R. Angus Silver, Padraig Gleeson, Ankur Sinha |
| Wellcome | 203048, 224499 | R. Angus Silver |
| Kavli Foundation | LS-2022-GR-40-2648 | Padraig Gleeson |
| Engineering and Physical Research Council | EP/X011151/1 | Padraig Gleeson |
| NIH | MH081905 | Sharon Crook |
| NIH | EB014640 | Sharon Crook |
| NIH | MH106674 | Sharon Crook |
| EU Horizon Europe | SEPTON (Gr. Agr. No. 101094901) | Sotirios Panagiotou |


**Figure 15.**

Documentation for the HindmarshRose1984Cell NeuroMLv2 ComponentType generated from the XSD schema and LEMS definitions on the NeuroML documentation website, showing its dynamics. More information about the ComponentType can be obtained from the tabs provided.

| Tool | Language/interface | Description | URL |
|---|---|---|---|
| pyNeuroML | Python/CLI | Recommended Python library for NeuroML; provides `pynml`, primary command line tool for NeuroML | docs.neuroml.org: pyNeuroML |
| libNeuroML | Python | Python API for NeuroML | docs.neuroml.org: LibNeuroML |
| NeuroMLlite | Python | High level library for creating NeuroML network models (beta) | docs.neuroml.org: NeuroMLlite |
| PyLEMS | Python | Python API and simulator for LEMS | docs.neuroml.org: PyLEMS |
| jLEMS | Java/CLI | Java API for LEMS and reference simulator | docs.neuroml.org: jLEMS |
| org.neuroml.model | Java | Java API for NeuroML | GitHub |
| org.neuroml.export | Java | Java API for translating NeuroML into different formats such as NEURON | GitHub |
| org.neuroml.import | Java | Java API for importing formats into LEMS and NeuroML | GitHub |
| jNeuroML | Java/CLI | Wraps jLEMS and all export/import packages and provides the `jnml` tool | docs.neuroml.org |
| NeuroML-C++ | C++ | C++ API for NeuroML | docs.neuroml.org: NeuroML C++ |
| NeuroML Toolbox | MATLAB | MATLAB NeuroML Toolbox | docs.neuroml.org: MATLAB |

## Appendix 1 Table 1

NeuroML software core tools, with a description of their scope, the main programming language they use (or other interaction means, e.g. Command Line Interface (CLI)), and links for more information.

| Tool | Language/interface | Description | URL |
|---|---|---|---|
| | | Simulation engines | |
| NEURON | Python/Hoc/CLI/GUI | Empirically-based simulations of neurons and networks of neurons | NEURON and NeuroML |
| NetPyNE | Python/web | Package to facilitate the development, parallel simulation, analysis, and optimization of biological neuronal networks using the NEURON simulator. Includes a graphical web interface | NetPyNE and NeuroML |
| EDEN | NeuroML | NeuroML-based neural simulator | EDEN and NeuroML |
| MOOSE | Python | The Multiscale Object-Oriented Simulation Environment is the base and numerical core for large, detailed multi-scale simulations that span computational neuroscience and systems biology. Based on a reimplementation of the GENESIS 2 core. | MOOSE and NeuroML |
| PyNN | Python | A simulator-independent language for building neuronal network models | PyNN and NeuroML |
| NEST | Python/SLI | Simulator for spiking neural network models focusing on dynamics, size, and structure of neural systems | NEST and NeuroML |
| Brian2 | Python | Easy to learn and use simulator for spiking neural networks | Brian2 and NeuroML |
| Arbor | Python | A multi-compartment neuron simulation library | Arbor and NeuroML |
| N2A | Java/GUI | Language and IDE for writing and simulating models | N2A and NeuroML |
| | | Databases | |
| OSB | Web | Resource for sharing and collaboratively developing computational models of neural systems | opensourcebrain.org |
| NeuroML-DB | Web | NeuroML database of cell and channel models | neuroml-db.org |
| | | Other tools | |
| OMV | Python | Open Source Brain Model Validation framework | OMV |
| SciUnit | Python | Data driven unit testing framework | sciunit.io |
| BluePyOpt | Python | Blue Brain Python Optimization Library | BluePyOpt |
| NeuroTune | Python | Package for fitting/optimization of NeuroML models | GitHub |
| PyElectro | Python | Electrophysiology analysis package | GitHub |

## Appendix 1 Table 2

Tools in the wider NeuroML software ecosystem, with a description of their scope, the main programming language they use (or other interaction means, e.g. through a web browser, Graphical User Interface (GUI) or Command Line Interface (CLI)), and links for more information.

| Test | Description |
|---|---|
| Schema tests | |
| Check names | Check that names of all elements, attributes, parameters match those provided in the schema |
| Check types | Check that the types of all included elements |
| Check values | Check that values follow given restrictions |
| Check inclusion | Check that required elements are included |
| Check cardinality | Check the number of elements |
| Check hierarchy | Check that child/children elements are included in the correct parent elements |
| Check sequence order | Check that child/children elements are included in the correct order |
| Additional tests | |
| Check top level ids | Check that top level (root) elements have unique ids |
| Check `Network` level ids | Check that child/children of the `Network` element have unique ids |
| Check `Cell` `Segment` ids | Check that all `Segment`s in a `Cell` have unique ids |
| Check single `Segment` without parent | Check that only one `Segment` is without parents (the soma `Segment`) |
| Check `SegmentGroup` ids | Check that all `SegmentGroup`s in a `Cell` have unique ids |
| Check `Member` segment ids exist | Check that `Segment`s referred to in `SegmentGroup` `Member`s exist |
| Check `SegmentGroup` definition | Check that `SegmentGroup`s being referenced are defined |
| Check `SegmentGroup` definition order | Check that `SegmentGroup`s are defined before being referenced |
| Check included `SegmentGroup`s | Check that `SegmentGroup`s referenced by `Include` elements of other `SegmentGroup`s exist |
| Check `numberInternalDivisions` | Check that `SegmentGroup`s define `numberInternalDivisions` (used by simulators to discretize un-branched branches into compartments for simulation) |
| Check included model files | Check that model files included by other files exist |
| Check `Population` component | Check that a component id provided to a `Population` exists |
| Check ion channel exists | Check that an ion channel used to define a `ChannelDensity` element exists |
| Check concentration model species | Check that the species used in `ConcentrationModel` elements are defined |
| Check `Population` size | Check that the `size` attribute of a `PopulationList` matches the number of defined `Instance`s |
| Check `Projection` component | Check that `Population`s used in the `Projection` elements exist |
| Check `Connection` `Segment` | Check that the `Segment` used in `Connection` elements exist |
| Check `Connection` pre/post cells | Check that the pre- and post-synaptic cells used in `Connection` elements exist and are correctly specified |
| Check `Synapse` | Check that the `Synapse` component used in a `Projection` element exists |
| Check root id | Check that the root `Segment` in a `Cell` morphology has id `0` |

## Appendix 1 Table 3

Listing of validation tests run by NeuroML

| Core components | | |
|---|---|---|
| annotation | bqbiol_encodes | bqbiol_hasPart |
| bqbiol_hasProperty | bqbiol_hasTaxon | bqbiol_hasVersion |
| bqbiol_is | bqbiol_isDescribedBy | bqbiol_isEncodedBy |
| bqbiol_isHomologTo | bqbiol_isPartOf | bqbiol_isPropertyOf |
| bqbiol_isVersionOf | bqbiol_occursIn | bqmodel_is |
| bqmodel_isDerivedFrom | bqmodel_isDescribedBy | rdf_Bag |
| rdf_Description | rdf_li | rdf_RDF |
| property | point3DWithDiam | notes |
| **Core dimensions** | | |
| area | capacitance | charge |
| charge_per_mole | concentration | conductance |
| conductance_per_voltage | conductanceDensity | current |
| currentDensity | idealGasConstantDims | length |
| per_time | per_voltage | permeability |
| resistance | resistivity | rho_factor |
| specificCapacitance | substance | temperature |
| time | voltage | volume |
| **Abstract cell models** | | |
| adExIaFCell | fitzHughNagumoCell | hindmarshRose1984Cell |
| iafCell | iafRefCell | iafTauCell |
| iafTauRefCell | izhikevich2007Cell | izhikevichCell |
| pinskyRinzelCA3Cell | | |
| **`ComponentTypes` related to biophysically detailed cells** | | |
| biophysicalProperties | biophysicalProperties2CaPools | cell |
| cell2CaPools | concentrationModel | decayingPoolConcentrationModel |
| distal | distalProperties | fixedFactorConcentrationModel |
| fixedFactorConcentrationModelTraub | from | include |
| inhomogeneousParameter | inhomogeneousValue | initMembPotential |
| intracellularProperties | intracellularProperties2CaPools | member |
| membraneProperties | membraneProperties2CaPools | morphology |
| parent | path | pointCellCondBased |
| pointCellCondBasedCa | proximal | proximalProperties |
| segment | segmentGroup | species |
| spikeThresh | subTree | to |
| variableParameter | channelDensity | channelDensityGHK |
| channelDensityGHK2 | channelDensityNernst | channelDensityNernstCa2 |
| channelDensityNonUniform | channelDensityNonUniformGHK | channelDensityNonUniformNernst |
| channelDensityVShift | channelPopulation | channelPopulationNernst |
| **`ComponentTypes` related to ion channels** | | |
| fixedTimeCourse | forwardTransition | gate |
| gateFractional | gateHHInstantaneous | gateHHrates |
| gateHHratesInf | gateHHratesTau | gateHHratesTauInf |
| gateHHtauInf | gateKS | HHExpLinearRate |
| HHExpLinearVariable | HHExpRate | HHExpVariable |
| HHSigmoidRate | HHSigmoidVariable | ionChannel |
| ionChannelHH | ionChannelKS | ionChannelPassive |
| ionChannelVShift | KSState | KSTransition |
| openState | q10ConductanceScaling | q10ExpTemp |
| q10Fixed | reverseTransition | subGate |
| tauInfTransition | vHalfTransition | closedState |

## Appendix 1 Table 4

Index of standard NeuroMLv2 ComponentTypes

| `ComponentTypes` related to synapses | | |
|---|---|---|
| alphaCurrentSynapse | alphaSynapse | blockingPlasticSynapse |
| doubleSynapse | expOneSynapse | expThreeSynapse |
| expTwoSynapse | gapJunction | gradedSynapse |
| linearGradedSynapse | silentSynapse | stdpSynapse |
| tsodyksMarkramDepFacMechanism | tsodyksMarkramDepMechanism | voltageConcDepBlockMechanism |
| `ComponentTypes` related to inputs | | |
| compoundInput | compoundInputDL | poissonFiringSynapse |
| pulseGenerator | pulseGeneratorDL | rampGenerator |
| rampGeneratorDL | sineGenerator | sineGeneratorDL |
| spike | spikeArray | spikeGenerator |
| spikeGeneratorPoisson | spikeGeneratorRandom | spikeGeneratorRefPoisson |
| timedSynapticInput | transientPoissonFiringSynapse | voltageClamp |
| voltageClampTriple | | |
| `ComponentTypes` related to networks | | |
| connection | connectionWD | continuousConnection |
| continuousConnectionInstance | continuousConnectionInstanceW | continuousProjection |
| electricalConnection | electricalConnectionInstance | electricalConnectionInstanceW |
| electricalProjection | explicitConnection | explicitInput |
| input | inputList | inputW |
| instance | location | network |
| networkWithTemperature | population | populationList |
| projection | rectangularExtent | region |
| synapticConnection | synapticConnectionWD | |
| `ComponentTypes` related to model simulation | | |
| Display | EventOutputFile | EventSelection |
| Line | OutputColumn | OutputFile |
| Simulation | | |
| `ComponentTypes` related to PyNN | | |
| alphaCondSynapse | alphaCurrSynapse | EIF_cond_alpha_isfa_ista |
| EIF_cond_exp_isfa_ista | expCondSynapse | expCurrSynapse |
| HH_cond_exp | IF_cond_alpha | IF_cond_exp |
| IF_curr_alpha | IF_curr_exp | SpikeSourcePoisson |

**Appendix 1 Table 5.**

Index of standard NeuroMLv2 ComponentTypes (continued)

| Model | Description | URL |
|---|---|---|
| **Neocortex** | | |
| ***Billeh et al.* (*2020*)** | Morphologically detailed and point neuron models based on electrophysiological recordings from visual cortex neurons | URL |
| ***Brunel* (*2000*)** | Spiking network illustrating balance between excitation and inhibition | URL |
| ***Hay et al.* (*2011*)** | Layer 5 pyramidal cell model constrained by somatic and dendritic recordings | URL |
| ***Izhikevich* (*2004*)** | Spiking neuron model reproducing wide range of neuronal activity | URL |
| ***Markram et al.* (*2015*)** | Cell models from Neocortical Microcircuit of Blue Brain Project | URL |
| ***Pospischil et al.* (*2008*)** | HH based models for different classes of cortical and thalamic neurons | URL |
| ***Potjans and Diesmann* (*2014*)** | Microcircuit model of sensory cortex with 8 populations across 4 layers | URL |
| ***Dura-Bernal et al.* (*2017*)** | Model of mouse primary motor cortex (M1) | URL |
| ***Sadeh et al.* (*2017*)** | Point neuron model of Inhibition Stabilized Network | URL |
| ***Smith et al.* (*2013*)** | Layer 2/3 cell model used to investigate dendritic spikes | URL |
| ***Traub et al.* (*2005*)** | Single column network model containing 14 cell populations from cortex and thalamus | URL |
| ***Bahl et al.* (*2012*)** | A set of reduced models of layer 5 pyramidal neurons | URL |
| ***Wilson and Cowan* (*1972*)** | A classic rate based model describing the dynamics and interactions between the excitatory and inhibitory populations of neurons | URL |
| ***Garcia del Molino et al.* (*2017*)** | Rate based model showing paradoxical response reversal of top-down modulation in cortical circuits with three interneuron types | URL |
| ***Mejias et al.* (*2016*)** | A rate based model simulating the dynamics of a cortical laminar structure across multiple scales: intralaminar, interlaminar, interareal and whole cortex | URL |
| **Cerebellum** | | |
| ***Maex and De Schutter* (*1998*)** | Cerebellar granule cell | URL |
| ***Cayco-Gajic et al.* (*2017*)** | Cerebellar granule cell layer network | URL |
| ***Maex and De Schutter* (*1998*)** | 3D Cerebellar granule cell layer network | URL |
| ***Solinas et al.* (*2007*)** | Cerebellar Golgi cell model | URL |
| ***Vervaeke et al.* (*2010*)** | Electrically connected cerebellar Golgi cell network model | URL |
| **Hippocampus** | | |
| ***Bezaire et al.* (*2016*)** | Full scale network model of CA1 region of hippocampus | URL |
| ***Ferguson et al.* (*2013*)** | Parvalbumin-positive interneuron from CA1, based on Izhikevich cell model | URL |
| ***Ferguson et al.* (*2014*)** | Pyramidal cell from CA1, based on Izhikevich cell model | URL |
| ***Migliore et al.* (*2005*)** | Multicompartmental model of pyramidal cell from CA1 region of hippocampus | URL |
| ***Pinsky and Rinzel* (*1994*)** | Simplified model of CA3 pyramidal cell | URL |
| ***Wang and Buzsáki* (*1996*)** | Hippocampal interneuronal network model exhibiting gamma oscillations | URL |
| **Olfactory bulb** | | |
| ***Migliore et al.* (*2014*)** | Large scale 3D olfactory bulb network with detailed mitral cells and granule cells | URL |
| **Invertebrate** | | |
| ***Hodgkin and Huxley* (*1952*)** | Classic investigation of the ionic basis of the action potential | URL |
| ***FitzHugh* (*1961*)** | Simplified form of Hodgkin Huxley model | URL |
| ***Prinz et al.* (*2004*)** | Pyloric network of the lobster stomatogastric ganglion system | URL |
| ***Boyle and Cohen* (*2008*)** | Model of body wall muscle from C. elegans | URL |
| ***Gleeson et al.* (*2018*)** | A multiscale framework for modelling the nervous system of C. elegans | URL |
| **General** | | |
| ***Morris and Lecar* (*1981*)** | Two dimensional reduced neuron model with calcium and potassium conductances | URL |
| ***Hindmarsh and Rose* (*1984*)** | A simplified point cell model which captures complex firing patterns of single neurons, such as periodic and chaotic bursting | URL |
| **Showcases** | | |
| NEST Showcase | Examples of interactions with simulator NEST | URL |
| PyNN Showcase | Examples of interactions between PyNN and NeuroML | URL |
| NetPyNE Showcase | Examples of interactions between NeuroML and NetPyNE | URL |
| SBML Showcase | Examples of interactions between NeuroML and SBML | URL |
| Brian Showcase | Examples of interactions between NeuroML and Brian | URL |
| MOOSE Showcase | Examples of interactions between NeuroML and MOOSE | URL |
| Arbor Showcase | Examples of interactions between NeuroML and Arbor | URL |
| EDEN Showcase | Examples of interactions between NeuroML and EDEN | URL |
| The Virtual Brain Showcase | Examples of interactions between NeuroML and TVB | URL |
| NEURON Showcase | Examples of interactions between NeuroML and NEURON | URL |
| neuroConstruct Showcase | Examples of neuroConstruct projects | URL |
| NeuroMorpho.Org | Examples of reconstructions from NeuroMorpho.Org | URL |
| Janelia MouseLight | Janelia MouseLight project neuronal reconstructions | URL |

## Appendix 1 Table 6

Listing of NeuroML models and example repositories

# References


Abi Akar N, Cumming B, Karakasis V, Küsters A, Klijn W, Peyser A, Yates S. (2019) **Arbor — A Morphologically-Detailed Neural Network Simulation Library for Contemporary High-Performance Computing Architectures** *In: 2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)* :274–282 https://doi.org/10.1109/EMPDP.2019.8671560

Abrams MB *et al.* (2022) **A Standards Organization for Open and FAIR Neuroscience: the International Neuroinformatics Coordinating Facility** *Neuroinformatics* **20**:25–36 https://doi.org/10.1007/s12021-020-09509-0

Ascoli GA, Donohue DE, Halavi M. (2007) **NeuroMorpho.org: a central resource for neuronal morphologies** *Journal of Neuro-science* **27**:9247–9251 https://doi.org/10.1523/JNEUROSCI.2055-07.2007

Awile O *et al.* (2022) **Modernizing the NEURON Simulator for Sustainability, Portability, and Performance** *Frontiers in Neuroinformatics* **16** https://doi.org/10.3389/fninf.2022.884046

Bahl A, Stemmler MB, Herz AVM, Roth A., special Issue on Computational Neuroscience (2012) **Automated optimization of a reduced layer 5 pyramidal cell model based on experimental data** *Journal of Neuroscience Methods* **210**:22–34 https://doi.org/10.1016/j.jneumeth.2012.04.006

Bergmann FT *et al.* (2014) **COMBINE archive and OMEX format: one file to share all information to reproduce a modeling project** *BMC Bioinformatics* **15** https://doi.org/10.1186/s12859-014-0369-z

Bezaire MJ, Raikov I, Burk K, Vyas D, Soltesz I. (2016) **Interneuronal mechanisms of hippocampal theta oscillation in a full-scale model of the rodent CA1 circuit** *eLife* **5**

Billeh YN *et al.* (2020) **Systematic Integration of Structural and Functional Data into Multi-scale Models of Mouse Primary Visual Cortex** *Neuron* **106**:388–403 https://doi.org/10.1016/j.neuron.2020.01.040

Billings G, Piasini E, Lőrincz A, Nusser Z, Silver RA (2014) **Network Structure within the Cerebellar Input Layer Enables Lossless Sparse Encoding** *Neuron* **83**:960–974 https://doi.org/10.1016/j.neuron.2014.07.020

Birgiolas J, Dietrich SW, Crook S, Rajadesingan A, Zhang C, Penchala SV, Addepalli V. (2015) **Ontology-Assisted Keyword Search for NeuroML Models** *In: Proceedings of the 27th International Conference on Scientific and Statistical Database Management SSDBM '15* https://doi.org/10.1145/2791347.2791360

Birgiolas J, Haynes V, Gleeson P, Gerkin RC, Dietrich SW, Crook S. (2023) **NeuroML-DB: Sharing and characterizing data-driven neuroscience models described in NeuroML** *PLOS Computational Biology* **19**:1–29 https://doi.org/10.1371/journal.pcbi.1010941

Blundell I *et al.* (2018) **Code Generation in Computational Neuroscience: A Review of Tools and Techniques** *Front Neuroinform* **12**

Bower JM, Beeman D. (1997) **The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System**

Boyle JH, Cohen N., seventh International Workshop on Information Processing in Cells and TissuesIPCAT 2007 (2008) **Caenorhabditis elegans body wall muscles are simple actuators** *Biosystems* **94**:170–181 https://doi.org/10.1016/j.biosystems.2008.05.025

Brunel N. (2000) **Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons** *Journal of computational neuroscience* **8**:183–208 https://doi.org/10.1023/A:1008925309027

Campagnola L *et al.* (2023) **Campagnola L, Larson E, Klein A, Hoese D, Siddharth Rossant C, Griffiths A, Rougier NP, asnt Gaifas L, Mühlbauer K, Taylor A MSS, Lambert T, sylm21, Champandard AJ, Hunter M, Robitaille T, Kaptan MF, de Andrade ES, et al., vispy/vispy: Version 0.13.0. Zenodo; 2023. 10.5281/zenodo.7945364, doi: 10.5281/zenodo.7945364.** https://doi.org/10.5281/zenodo.7945364

Cannon RC *et al.* (2007) **Interoperability of Neuroscience Modeling Software: Current Status and Future Directions** *Neuroinformatics* **5**:127–138 https://doi.org/10.1007/s12021-007-0004-5

Cannon RC, Gleeson P, Crook S, Ganapathy G, Marin B, Piasini E, Silver RA (2014) **LEMS: a language for expressing complex biological models in concise and hierarchical form and its use in underpinning NeuroML 2** *Frontiers in Neuroinformatics* **8** https://doi.org/10.3389/fninf.2014.00079

Cayco-Gajic NA, Clopath C, Silver RA (2017) **Sparse synaptic connectivity is required for decorrelation and pattern separation in feedforward networks** *Nat Commun* **8**

Choi K, Medley JK, König M, Stocking K, Smith L, Gu S, Sauro HM (2018) **Tellurium: An extensible python-based modeling environment for systems and synthetic biology** *Biosystems* **171**:74–79 https://doi.org/10.1016/j.biosystems.2018.07.006

Dai K *et al.* (2020) **The SONATA data format for efficient description of large-scale network models** *PLOS Computational Biology* **16**:1–24 https://doi.org/10.1371/journal.pcbi.1007696

Davison AP, Brüderle D, Eppler J, Kremkow J, Muller E, Pecevski D, Perrinet L, Yger P. (2009) **PyNN: A Common Interface for Neuronal Network Simulators** *Front Neuroinform* **2**

De Schutter E, Bower JM (1994) **An active membrane model of the cerebellar Purkinje cell** *I. Simulation of current clamps in slice. Journal of Neurophysiology* **71**:375–400

Denker M, Yegenoglu A, Grün S. (2018) **Collaborative HPC-enabled workflows on the HBP Collaboratory using the Elephant framework** https://doi.org/10.12751/incf.ni2018.0019

Druckmann S, Banitt Y, Gidon A, Schürmann F, Markram H, Segev I. (2007) **A novel multiple objective optimization framework for constraining conductance-based neuron models by experimental data** *Frontiers in Neuroscience* **1** https://doi.org/10.3389/neuro.01.1.1.001.2007

Dura-Bernal S, Neymotin SA, Kerr CC, Sivagnanam S, Majumdar A, Francis JT, Lytton WW (2017) **Evolutionary algorithm optimization of biological learning parameters in a biomimetic neuroprosthesis** *IBM J Res Dev* **61**:6–6

Dura-Bernal S *et al.* (2019) **NetPyNE, a tool for data-driven multiscale modeling of brain circuits** *Elife* **8** https://doi.org/10.7554/elife.44494

Einevoll GT, Destexhe A, Diesmann M, Grün S, Jirsa V, de Kamps M, Migliore M, Ness TV, Plesser HE, Schürmann F. (2019) **The scientific case for brain simulations** *Neuron* **102**:735–744 https://doi.org/10.1016/j.neuron.2019.03.027

Executable Books Community (2020) **Jupyter Book** https://doi.org/10.5281/zenodo.4539666

Ferguson K, Huh C, Amilhon B, Williams S, Skinner F. (2014) **Simple, biologically-constrained CA1 pyramidal cell models using an intact, whole hippocampus context** *F1000Research* **3** https://doi.org/10.12688/f1000research.3894.1

Ferguson KA, Huh CY, Amilhon B, Williams S, Skinner FK (2013) **Experimentally constrained CA1 fast-firing parvalbumin-positive interneuron network models exhibit sharp transitions into coherent high frequency rhythms** *Frontiers in Computational Neuroscience* **7** https://doi.org/10.3389/fncom.2013.00144

FitzHugh R. (1961) **Impulses and Physiological States in Theoretical Models of Nerve Membrane** *Biophysical Journal* **1**:445–466

Garcia S *et al.* (2014) **Neo: an object model for handling electrophysiology data in multiple formats** *Frontiers in Neuroinformatics* **8** https://doi.org/10.3389/fninf.2014.00010

Gerkin RC, Birgiolas J, Jarvis RJ, Omar C, Crook SM (2019) **NeuronUnit: A package for data-driven validation of neuron models using SciUnit** https://doi.org/10.1101/665331

Gewaltig MO, Diesmann M. (2007) **NEST (NEural Simulation Tool)** *Scholarpedia J* **2**

Gleeson P, Steuber V, Silver RA (2007) **neuroConstruct: a tool for modeling networks of neurons in 3D space** *Neuron* **54**:219–235

Gleeson P *et al.* (2019) **Open Source Brain: A Collaborative Resource for Visualizing, Analyzing, Simulating, and Developing Standardized Models of Neurons and Circuits** *Neuron* **103**:395–411 https://doi.org/10.1016/j.neuron.2019.05.019

Gleeson P *et al.* (2010) **NeuroML: A Language for Describing Data Driven Models of Neurons and Networks with a High Degree of Biological Detail** *PLoS Computational Biology* **6** https://doi.org/10.1371/journal.pcbi.1000815

Gleeson P, Crook S, Turner D, Mantel K, Raunak M, Willke T, Cohen JD (2023) **Integrating model development across computational neuroscience, cognitive science, and machine learning** *Neuron* **111**:1526–1530 https://doi.org/10.1016/j.neuron.2023.03.037

Gleeson P, Lung D, Grosu R, Hasani R, Larson SD (2018) **c302: a multiscale framework for modelling the ner-vous system of caenorhabditis elegans** *Philosophical Transactions of the Royal Society B: Biological Sciences* **373** https://doi.org/10.1098/rstb.2017.0379

Goddard NH, Hucka M, Howell F, Cornelis H, Shankar K, Beeman D. (2001) **Towards NeuroML: Model Description Methods for Collaborative Modelling in Neuroscience** *Philosophical Transactions of the Royal Society of London Series B: Biological Sciences* **356**:1209–1228 https://doi.org/10.1098/rstb.2001.0910

Gorgolewski KJ *et al.* (2016) **The brain imaging data structure, a format for organizing and describing outputs of neuroimaging experiments** *Sci Data* **3**

Gurnani H, Silver RA (2021) **Multidimensional population activity in an electrically coupled inhibitory circuit in the cerebellar cortex** *Neuron* https://doi.org/10.1016/j.neuron.2021.03.027

Harris CR *et al.* (2020) **Array programming with NumPy** *Nature* **585**:357–362 https://doi.org/10.1038/s41586-020-2649-2

Hay E, Hill S, Schürmann F, Markram H, Segev I. (2011) **Models of Neocortical Layer 5b Pyramidal Cells Capturing a Wide Range of Dendritic and Perisomatic Active Properties** *PLoS Comput Biol* **7**

Hindmarsh JL, Rose RM (1984) **A Model of Neuronal Bursting Using Three Coupled First Order Differential Equations** *Proceedings of the Royal Society of London Series B* **221**:87–102 https://doi.org/10.1098/rspb.1984.0024

Hindmarsh JL, Rose RM, Huxley AF (1984) **A model of neuronal bursting using three coupled first order differential equations** *Proceedings of the Royal Society of London Series B Biological Sciences* **221**:87–102 https://doi.org/10.1098/rspb.1984.0024

Hines ML, Carnevale NT (1997) **The NEURON simulation environment** *Neural Computation* **9**:1179–1209

Hodgkin AL, Huxley AF (1952) **A quantitative description of membrane current and its application to conduction and excitation in nerve** *J Physiol* **117**:500–544

Hucka M *et al.* (2003) **The Systems Biology Markup Language (SBML): a medium for representation and exchange of biochemical network models** *Bioinformatics* **19**:524–531 https://doi.org/10.1093/bioinformatics/btg015

Hucka M *et al.* (2015) **Promoting coordinated development of community-based information standards for modeling in biology: the COMBINE initiative** *Frontiers in Bioengineering and Biotechnology* **3** https://doi.org/10.3389/fbioe.2015.00019

Hunter JD (2007) **Matplotlib: A 2D graphics environment** *Computing in Science & Engineering* **9**:90–95 https://doi.org/10.1109/MCSE.2007.55

INCF (2023) **INCF, Role of community standards; 2023. https://www.incf.org/role-community-standards, accessed: 2023-11-09.**

Izhikevich EM (2004) **Which model to use for cortical spiking neurons?** *IEEE transactions on neural networks* **15**:1063–1070 https://doi.org/10.1109/tnn.2004.832719

Kriener B, Hu H, Vervaeke K. (2022) **Parvalbumin interneuron dendrites enhance gamma oscillations** *Cell Reports* **39** https://doi.org/10.1016/j.celrep.2022.110948

Lapicque L. (1907) **Recherches quantitatives sur l'excitation électrique des nerfs traitée comme une polarisation** *J Physiol Pathol Gen* **9**:620–635

Larson SD, Martone ME (2013) **NeuroLex.org: an online framework for neuroscience knowledge** *Frontiers in neuroinfor-matics* **7**

Lloyd CM, Halstead MDB, Nielsen PF (2004) **CellML: its future, present and past** *Prog Biophys Mol Biol* **85**:433–450

Maex R, De Schutter E. (1998) **Synchronization of Golgi and Granule Cell Firing in a Detailed Network Model of the Cerebellar Granule Cell Layer** *Journal of Neurophysiology* **80**:2521–2537

Markram H *et al.* (2015) **Reconstruction and Simulation of Neocortical Microcir-cuitry** *Cell* **163**:456–492 https://doi.org/10.1016/j.cell.2015.09.029

Martone M, Das S, Goscinski W, Hellgren-Kotaleski J, Ho ETW, Kennedy D, Leergaard T, Wachtler T, Yamaguchi Y, Abrams M (2019) **Call for community review of NeuroML — A Model Description Language for Computational Neuroscience** *F1000 Research Limited* https://doi.org/10.7490/F1000RESEARCH.1116398.1

McDougal RA, Morse TM, Carnevale T, Marenco L, Wang R, Migliore M, Miller PL, Shepherd GM, Hines ML (2017) **Twenty years of ModelDB and beyond: building essential modeling tools for the future of neuroscience** *J Comput Neurosci* **42**:1–10

Mejias JF, Murray JD, Kennedy H, Wang XJ (2016) **Feedforward and feedback frequency-dependent interactions in a large-scale laminar network of the primate cortex** *Science Advances* **2** https://doi.org/10.1126/sciadv.1601335

Migliore M, Cavarretta F, Hines ML, Shepherd GG (2014) **Distributed organization of a brain microcircuit analysed by three-dimensional modeling: the olfactory bulb** *Frontiers in Computational Neuroscience* **8**

Migliore M, Ferrante M, Ascoli GA (2005) **Signal Propagation in Oblique Dendrites of CA1 Pyramidal Cells** *Journal of Neurophysiology* **94**:4145–4155

Migliore M, Morse TM, Davison AP, Marenco L, Shepherd GM, Hines ML (2003) **ModelDB: making models publicly accessible to support computational neuroscience** *Neuroinform* https://doi.org/10.1385/NI:1:1:135

Garcia del Molino LC, Yang GR, Mejias JF, Wang XJ (2017) **Paradoxical response reversal of top-down modulation in cortical circuits with three interneuron types** *eLife* **6** https://doi.org/10.7554/eLife.29742

Morris C, Lecar H. (1981) **Voltage oscillations in the barnacle giant muscle fiber** *Biophysical Journal* **35**:193–213 https://doi.org/10.1016/S0006-3495(81)84782-0

Muller E, Bednar JA, Diesmann M, Gewaltig MO, Hines M, Davison AP (2015) **Python in neuroscience** *Frontiers in Neuroin-formatics* **9** https://doi.org/10.3389/fninf.2015.00011

Neal ML *et al.* (2018) **Harmonizing semantic annotations for computational models in biology** *Briefings in Bioinformatics* **20**:540–550 https://doi.org/10.1093/bib/bby087

Omar C, Aldrich J, Gerkin RC (2014) **Collaborative Infrastructure for Test-Driven Scientific Model Validation** *In: Companion Proceedings of the 36th International Conference on Software Engineering ICSE Companion 2014* :524–527 https://doi.org/10.1145/2591062.2591129

Panagiotou S, Sidiropoulos H, Soudris D, Negrello M, Strydis C. (2022) **EDEN: A High-Performance, General-Purpose, NeuroML-Based Neural Simulator** *Frontiers in neuroinformatics* **16**

Pinsky PF, Rinzel J. (1994) **Intrinsic and network rhythmogenesis in a reduced traub model for CA3 neurons** *Journal of Computational Neuroscience* **1**:39–60 https://doi.org/10.1007/BF00962717

Poirazi P, Papoutsi A. (2020) **Illuminating dendritic function with computational models** *Nature Reviews Neuroscience* https://doi.org/10.1038/s41583-020-0301-7

Pospischil M, Toledo-Rodriguez M, Monier C, Piwkowska Z, Bal T, Frégnac Y, Markram H, Destexhe A. (2008) **Minimal Hodgkin-Huxley type models for different classes of cortical and thalamic neurons** *Biological Cybernetics* **99**:427–441 https://doi.org/10.1007/s00422-008-0263-8

Potjans TC, Diesmann M. (2014) **The Cell-Type Specific Cortical Microcircuit: Relating Structure and Activity in a Full-Scale Spiking Network Model** *Cereb Cortex* **24**:785–806

Prinz AA, Bucher D, Marder E. (2004) **Similar network activity from disparate circuit parameters** *Nature Neuroscience* **7**:1345–1352 https://doi.org/10.1038/nn1352

Jupyter Project *et al.* (2018) **Binder 2.0 - Reproducible, interactive, sharable environments for science at scale** *Proceedings of the 17th Python in Science Conference* :113–120 https://doi.org/10.25080/Majora-4af1f417-011

Rall W. (1962) **Theory of physiological properties of dendrites** *Annals of the New York Academy of Sciences* **96**:1071–1092

Ranjan R, Khazen G, Gambazzi L, Ramaswamy S, Hill S, Schürmann F, Markram H. (2011) **Channelpedia: an integrative and interactive database for ion channels** *Frontiers in Neuroinformatics* **5** https://doi.org/10.3389/fninf.2011.00036

Ray S, Bhalla US (2008) **PyMOOSE: interoperable scripting in Python for MOOSE** *Frontiers in Neuroinformatics* **2**

Rossant C, Goodman DF, Fontaine B, Platkiewicz J, Magnusson A, Brette R. (2011) **Fitting Neuron Models to Spike Trains** *Frontiers in Neuroscience* **5** https://doi.org/10.3389/fnins.2011.00009

Rothganger F, Warrender C, Trumbo D, Aimone J. (2014) **N2A: a computational tool for modeling from neurons to algorithms** *Frontiers in Neural Circuits* **8** https://doi.org/10.3389/fncir.2014.00001

Sadeh S, Silver RA, Mrsic-Flogel TD, Muir DR (2017) **Assessing the Role of Inhibition in Stabilizing Neocortical Networks Requires Large-Scale Perturbation of the Inhibitory Population** *The Journal of Neuroscience* **37**:12050–12067 https://doi.org/10.1523/jneurosci.0963-17.2017

Shaikh B *et al.* (2022) **BioSimulators: a central registry of simulation engines and services for recommending specific tools** *Nucleic Acids Research* https://doi.org/10.1093/nar/gkac331

Sinha A *et al.* (2023) **NeuroML/pyNeuroML** https://doi.org/10.5281/zenodo.8366699

Sinha A *et al.* (2023) **NeuralEnsemble/libNeuroML: v0.5.5** https://doi.org/10.5281/zenodo.8364786

Sivagnanam S, Majumdar A, Yoshimoto K, Astakhov V, Bandrowski AE, Martone ME, Carnevale NT, et al. (2013) **Introducing the neuroscience gateway** *IWSG* **993**

Smith SL, Smith IT, Branco T, Hausser M. (2013) **Dendritic spikes enhance stimulus selectivity in cortical neurons in vivo** *Nature* **503**:115–120

Solinas S, Forti L, Cesana E, Mapelli J, De Schutter E, D'Angelo E. (2007) **Computational reconstruction of pacemaking and intrinsic electroresponsiveness in cerebellar Golgi cells** *Frontiers in Cellular Neuroscience* **1**

Stimberg M, Brette R, Goodman DF (2019) **Brian 2, an intuitive and efficient neural simulator** *eLife* **8** https://doi.org/10.7554/eLife.47314

Teeters JL *et al.* (2015) **Neurodata Without Borders: Creating a Common Data Format for Neurophysiology** *Neuron* **88**:629–634

Traub RD, Contreras D, Cunningham MO, Murray H, LeBeau FEN, Roopun A, Bibbig A, Wilent WB, Higley MJ, Whittington MA (2005) **Single-Column Thalamocortical Network Model Exhibiting Gamma Oscillations, Sleep Spindles, and Epileptogenic Bursts** *Journal of Neurophysiology* **93**:2194–2232 https://doi.org/10.1152/jn.00983.2004

Van Geit W, Gevaert M, Chindemi G, Rössert C, Courcol JD, Muller EB, Schürmann F, Segev I, Markram H. (2016) **BluePyOpt: Leveraging open source software and cloud infrastructure to optimise model parameters in neuroscience** *Frontiers in Neuroinformatics* **10** https://doi.org/10.3389/fninf.2016.00017

Vella M, Cannon RC, Crook S, Davison AP, Ganapathy G, Robinson HPC, Silver RA, Gleeson P. (2014) **libNeuroML and PyLEMS: using Python to combine procedural and declarative modeling approaches in computational neuroscience** *Frontiers in neuroinformatics* **8** https://doi.org/10.3389/fninf.2014.00038

Vervaeke K, Lőrincz A, Gleeson P, Farinella M, Nusser Z, Silver RA (2010) **Rapid desynchronization of an electrically coupled interneuron network with sparse excitatory synaptic input** *Neuron* **67**:435–51

Waltemath D *et al.* (2011) **Reproducible computational biology experiments with SED-ML - The Simulation Experiment Description Markup Language** *BMC Systems Biology* **5** https://doi.org/10.1186/1752-0509-5-198

Wang XJ, Buzsáki G. (1996) **Gamma Oscillation by Synaptic Inhibition in a Hippocampal Interneuronal Network Model** *Journal of Neuroscience* **16**:6402–6413

Wilkinson MD *et al.* (2016) **The FAIR Guiding Principles for scientific data management and stewardship** *Scientific Data* **3** https://doi.org/10.1038/sdata.2016.18

Wilson HR, Cowan JD (1972) **Excitatory and inhibitory interactions in localized populations of model neurons** *Biophys J* **12**:1–24

Yao HK, Guet-McCreight A, Mazza F, Moradi Chameh H, Prevot TD, Griffiths JD, Tripathy SJ, Valiante TA, Sibille E, Hay E. (2022) **Reduced inhibition in depression impairs stimulus processing in human cortical microcircuits** *Cell Reports* **38** https://doi.org/10.1016/j.celrep.2021.110232

## Article and author information


**Ankur Sinha**
Department of Neuroscience, Physiology and Pharmacology, University College London, United Kingdom
ORCID iD: 0000-0001-7568-7167

**Padraig Gleeson**
Department of Neuroscience, Physiology and Pharmacology, University College London, United Kingdom
**For correspondence:** p.gleeson@ucl.ac.uk
ORCID iD: 0000-0001-5963-8576

**Bóris Marin**
Universidade Federal do ABC, São Bernardo do Campo, Brazil
ORCID iD: 0000-0001-5503-648X

**Salvador Dura-Bernal**
State University of New York, Brooklyn, USA, Center for Biomedical Imaging and Neuromodulation, Nathan S. Kline Institute for Psychiatric Research, Orangeburg, NY, USA
ORCID iD: 0000-0002-8561-5324

**Sotirios Panagiotou**
Erasmus Medical Center, Rotterdam, Netherlands

**Sharon Crook**
Arizona State University, USA
ORCID iD: 0000-0003-1659-0749

**Matteo Cantarelli**
MetaCell Ltd.
ORCID iD: 0000-0002-0054-226X

**Robert C. Cannon**
Opus2 International Ltd, UK
ORCID iD: 0000-0003-4259-4460

**Andrew P. Davison**
CNRS, Gif-sur-Yvette, France
ORCID iD: 0000-0002-4793-7541

**Harsha Gurnani**
Department of Neuroscience, Physiology and Pharmacology, University College London, United Kingdom, University of Washington, Seattle, USA
ORCID iD: 0000-0001-6383-3104

**R. Angus Silver**
Department of Neuroscience, Physiology and Pharmacology, University College London, United Kingdom
**For correspondence:** a.silver@ucl.ac.uk
ORCID iD: 0000-0002-5480-6638

## Copyright



## Editors

Reviewing Editor
**Eilif Muller**
University of Montreal, Montreal, Canada

Senior Editor
**Panayiota Poirazi**
FORTH Institute of Molecular Biology and Biotechnology, Heraklion, Greece

**Reviewer #1 (Public Review):**

Summary:

The manuscript gives a broad overview of how to write NeuroML, and a brief description of how to use it with different simulators and for different purposes - cells to networks, simulation, optimization, and analysis. From this perspective, it can be an extremely useful document to introduce new users to NeuroML.

However, the manuscript itself seems to lose sight of this goal in many places, and instead, the description at times seems to target software developers. For example, there is a long paragraph on the board and user community. The discussion on simulator tools seems more for developers, not users. All the information presented at the level of a developer is likely to be distracting to readers..

Strengths:

The modularity of NeuroML is indeed a great advantage. For example, the ability to specify the channel file allows different channels to be used with different morphologies without redundancy. The hierarchical nature of NeuroML also is commendable, and well illustrated in Figures 2a through c.

The number of tools available to work with NeuroML is impressive.

The abstract, beginning, and end of the manuscript present and discuss incorporating NeuroML into research workflows to support FAIR principles.

Having a Python API and providing examples using this API is fantastic. Exporting to NeuroML from Python is also a great feature.

Weaknesses:

Though modularity is a strength, it is unclear to me why the cell morphology isn't also treated similarly, i.e., specify the morphology of a multi-compartmental model in a separate file, and then allow the cell file to specify not only the files containing channels, but also the file containing the multi-compartmental morphology, and then specify the conductance for different segment groups. Also, after pynml_write_neuroml2_file, you would not have a super long neuroML file for each variation of conductances, since there would be no need to rewrite the multi-compartmental morphology for each conductance variation.

This would be especially important for optimizations, if each trial optimization wrote out the neuroML file, then including the full morphology of a realistic cell would take up excessive disk space, as opposed to just writing out the conductance densities. As long as cell morphology must be included in every cell file, then NeuroML is not sufficiently modular, and the authors should moderate their claim of modularity (line 419) and building blocks (551). In addition, this is very important for downloading NeuroML-compliant reconstructions from NeuroMorpho.org. If the cell morphology cannot be imported, then the user has to edit the file downloaded from NeuroMorpho.org, and provenance can be lost. Also, Figure 2d loses the hierarchical nature by showing ion channels, synapses, and networks as separate main branches of NeuroML.

In Figure 5, the difference between the core and native simulator is unclear. What is involved in helper scripts? I thought neurons could read NeuroML? If so, why do you need the export simulator-specific scripts? In addition, it seems strange to call something the "core" simulation engine, when it cannot support multi-compartmental models. It is unclear why "other simulators" that natively support NeuroML cannot be called the core. It might be more helpful to replace this sort of classification with a user-targeted description. The authors already state which simulators support NeuroML and which ones need code to be exported. In contrast, lines 369-370 mention that not all NeuroML models are supported by each simulator. I recommend expanding this to explain which features are supported in each simulator. Then, the unhelpful separation between core and native could be eliminated.

The body of the manuscript has so much other detail that I lose sight of how NeuroML supports FAIR. It is also unclear who is the intended audience. When I get to lines 336-344, it seems that this description is too much detail for the audience. The paragraph beginning on line 691 is a great example of being unclear about who is the audience. Does someone wanting to develop NeuroML models need to understand XSD schema? If so, the explanation is not clear. XSD schema is not defined and instead explains NeuroML-specific aspects of XSD. Lines 734-735 are another example of explaining to code developers (not model developers).

https://doi.org/10.7554/eLife.95135.1.sa1

**Reviewer #2 (Public Review):**

Summary:

Developing neuronal models that are shareable, reproducible, and interoperable allows the neuroscience community to make better use of published models and to collaborate more effectively. In this manuscript, the authors present a consolidated overview of the NeuroML model description system along with its associated tools and workflows. They describe where different components of this ecosystem lay along the model development pathway and highlight resources, including documentation and tutorials, to help users employ this system.

Strengths:

The manuscript is well-organized and clearly written. It effectively uses the delineated model development life cycle steps, presented in Figure 1, to organize its descriptions of the

different components and tools relating to NeuroML. It uses this framework to cover the breadth of the software ecosystem and categorize its various elements. The NeuroML format is clearly described, and the authors outline the different benefits of its particular construction. As primarily a means of describing models, NeuroML also depends on many other software components to be of high utility to computational neuroscientists; these include simulators (ones that both pre-date NeuroML and those developed afterwards), visualization tools, and model databases.

Overall, the rationale for the approach NeuroML has taken is convincing and well-described. The pointers to existing documentation, guides, and the example usages presented within the manuscript are useful starting points for potential new users. This manuscript can also serve to inform potential users of features or aspects of the ecosystem that they may have been unaware of, which could lower obstacles to adoption. While much of what is presented is not new to this manuscript, it still serves as a useful resource for the community looking for information about an established, but perhaps daunting, set of computational tools.

Weaknesses:

The manuscript in large part catalogs the different tools and functionalities that have been produced through the long development cycle of NeuroML. As discussed above, this is quite useful, but it can still be somewhat overwhelming for a potential new user of these tools. There are new user guides (e.g., Table 1) and example code (e.g. Box 1), but it is not clear if those resources employ elements of the ecosystem chosen primarily for their didactic advantages, rather than general-purpose utility. I feel like the manuscript would be strengthened by the addition of clearer recommendations for users (or a range of recommendations for users in different scenarios).

For example, is the intention that most users should primarily use the core NeuroML tools and expand into the wider ecosystem only under particular circumstances? What are the criteria to keep in mind when making that decision to use alternative tools (scale/complexity of model, prior familiarity with other tools, etc.)? The place where it seems most ambiguous is in the choice of simulator (in part because there seem to be the most options there) - are there particular scenarios where the authors may recommend using simulators other than the core jNeuroML software?

The interoperability of NeuroML is a major strength, but it does increase the complexity of choices facing users entering into the ecosystem. Some clearer guidance in this manuscript could enable computational neuroscientists with particular goals in mind to make better strategic decisions about which tools to employ at the outset of their work.

https://doi.org/10.7554/eLife.95135.1.sa0