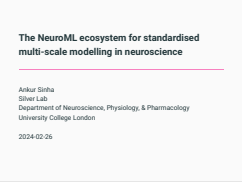
The NeuroML ecosystem for standardised multi-scale modelling in neuroscience

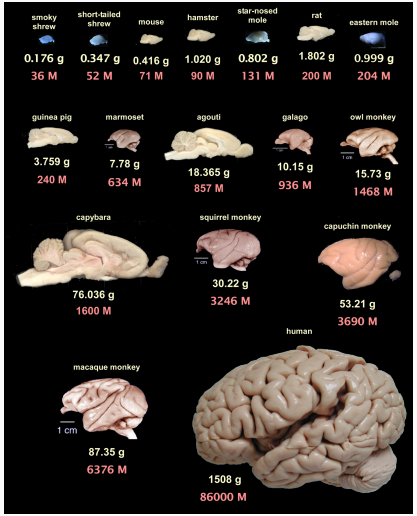
Ankur Sinha
Silver Lab
Department of Neuroscience, Physiology, & Pharmacology
University College London

2024-02-26

2024-02-27
The NeuroML ecosystem for standardised multi-scale modelling in neuroscience



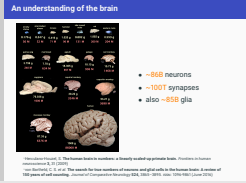
An understanding of the brain



- ~86B neurons
- ~100T synapses
- also ~85B glia

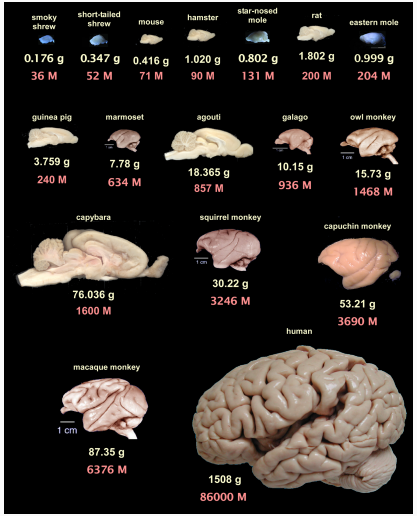
[1] Herculano-Houzel, S. **The human brain in numbers: a linearly scaled-up primate brain.** *Frontiers in human neuroscience* **3,** 31 (2009)

[1] von Bartheld, C. S. *et al.* **The search for true numbers of neurons and glial cells in the human brain: A review of 150 years of cell counting.** *Journal of Comparative Neurology* **524,** 3865–3895. ISSN: 1096-9861 (June 2016)



1. The most recent estimate puts the number of neurons in the human brain at 86B.

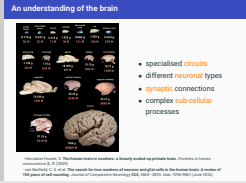
An understanding of the brain



- specialised circuits
- different neuronal types
- synaptic connections
- complex sub-cellular processes

[1] Herculano-Houzel, S. **The human brain in numbers: a linearly scaled-up primate brain.** *Frontiers in human neuroscience* **3,** 31 (2009)

[1] von Bartheld, C. S. *et al.* **The search for true numbers of neurons and glial cells in the human brain: A review of 150 years of cell counting.** *Journal of Comparative Neurology* **524,** 3865–3895. ISSN: 1096-9861 (June 2016)



1. The most recent estimate puts the number of neurons in the human brain at 86B.

Experiments provide a window into the brain

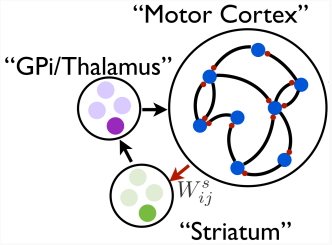
Multiple scales of experiments/data sources go here

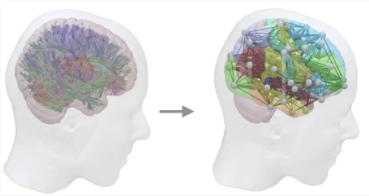
Experiments provide a window into the brain

Multiple scales of experiments/data sources go here

1. There is so much data out there now, as we embrace Open Science.

Models test & unify experimental results; generate hypotheses



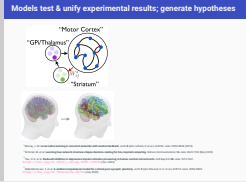


[1] Murray, J. M. **Local online learning in recurrent networks with random feedback.** *eLife* **8** (eds Latham, P. *et al.*) e43299. ISSN: 2050-084X (2019)

[1] Schirner, M. *et al.* **Learning how network structure shapes decision-making for bio-inspired computing.** *Nature Communications* **14.** ISSN: 2041-1723 (May 2023)

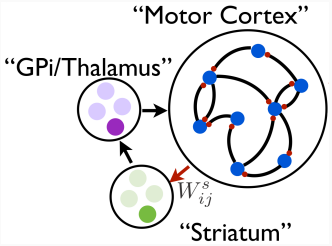
[1] Yao, H. K. *et al.* **Reduced inhibition in depression impairs stimulus processing in human cortical microcircuits.** *Cell Reports* **38.** ISSN: 2211-1247. https://doi.org/10.1016/j.celrep.2021.110232 (Jan. 2022)

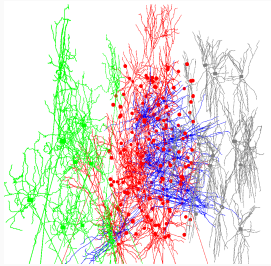
[1] Mäki-Marttunen, T. *et al.* **A unified computational model for cortical post-synaptic plasticity.** *eLife* **9** (eds Shouval, H. Z. *et al.*) e55714. ISSN: 2050-084X. https://doi.org/10.7554/eLife.55714 (July 2020)

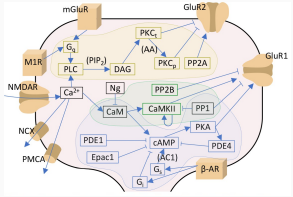


1. Models/theory are necessary for:
2. combining independent experimental results into unified theories
3. exploring these complex systems across wider range of conditions
4. generating new testable hypotheses
5. RNNs are appropriate for lots of projects, for example.
6. So are whole brain neural mass models.
7. But, to really understand the underlying mechanisms that give rise to emergent behaviour, we must model the brain at biophysically detailed levels.

Models test & unify experimental results; generate hypotheses





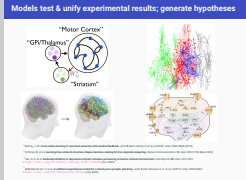


[1] Murray, J. M. **Local online learning in recurrent networks with random feedback.** *eLife* **8** (eds Latham, P. *et al.*) e43299. ISSN: 2050-084X (2019)

[1] Schirner, M. *et al.* **Learning how network structure shapes decision-making for bio-inspired computing.** *Nature Communications* **14.** ISSN: 2041-1723 (May 2023)

[1] Yao, H. K. *et al.* **Reduced inhibition in depression impairs stimulus processing in human cortical microcircuits.** *Cell Reports* **38.** ISSN: 2211-1247. https://doi.org/10.1016/j.celrep.2021.110232 (Jan. 2022)

[1] Mäki-Marttunen, T. *et al.* **A unified computational model for cortical post-synaptic plasticity.** *eLife* **9** (eds Shouval, H. Z. *et al.*) e55714. ISSN: 2050-084X. https://doi.org/10.7554/eLife.55714 (July 2020)

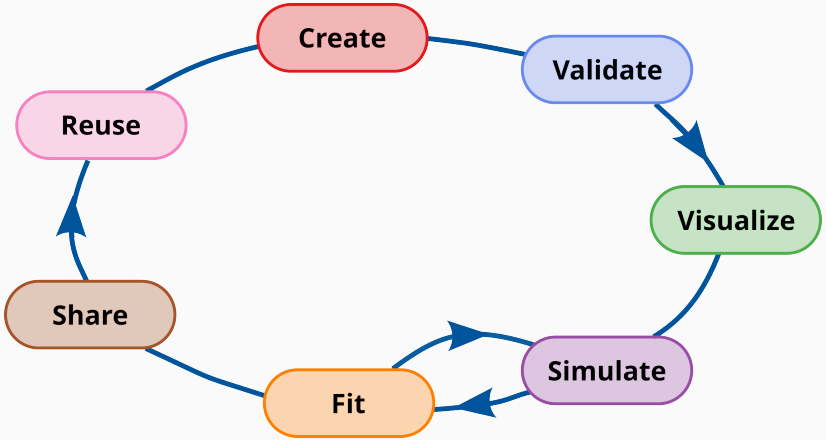


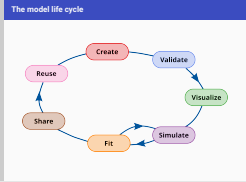
1. Models/theory are necessary for:
2. combining independent experimental results into unified theories
3. exploring these complex systems across wider range of conditions
4. generating new testable hypotheses
5. RNNs are appropriate for lots of projects, for example.
6. So are whole brain neural mass models.
7. But, to really understand the underlying mechanisms that give rise to emergent behaviour, we must model the brain at biophysically detailed levels.

A *mechanistic* understanding of the brain requires biophysically detailed modelling

A *mechanistic* understanding of the brain requires biophysically detailed modelling

The model life cycle





1. Ideally, what we want is for all the stages to be connected seamlessly, but this is not true in practice.
2. We create our model, ideally re-using already published components.
3. Then before we simulate our model, we want to validate it in some way.
4. We also want to analyse and visualise our model description before.
5. Then we iteratively simulate and fit our model to data, or to produce a certain behaviour.
6. Finally, we want to publish and openly share the model so others can use it in the future.

Computational modelling software ecosystem is fragmented

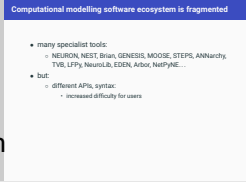
- many specialist tools:
  - NEURON, NEST, Brian, GENESIS, MOOSE, STEPS, ANNarchy, TVB, LFPy, NeuroLib, EDEN, Arbor, NetPyNE…

Computational modelling software ecosystem is fragmented
- many specialist tools:
  - NEURON, NEST, Brian, GENESIS, MOOSE, STEPS, ANNarchy, TVB, LFPy, NeuroLib, EDEN, Arbor, NetPyNE…

1. There are a lot of software tools out there for users to pick from, for different levels of modelling, optimisation, analysis.
2. For each stage.
3. But, they aren't designed to work together.
4. They have their own designs, their own APIs, syntax, model representation, and usually their own suite of custom utilities to work with their model representation.

Computational modelling software ecosystem is fragmented

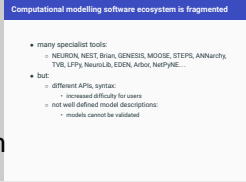
- many specialist tools:
  - NEURON, NEST, Brian, GENESIS, MOOSE, STEPS, ANNarchy, TVB, LFPy, NeuroLib, EDEN, Arbor, NetPyNE…
- but:
  - different APIs, syntax:
    - increased difficulty for users



1. There are a lot of software tools out there for users to pick from, for different levels of modelling, optimisation, analysis.
2. For each stage.
3. But, they aren't designed to work together.
4. They have their own designs, their own APIs, syntax, model representation, and usually their own suite of custom utilities to work with their model representation.

Computational modelling software ecosystem is fragmented

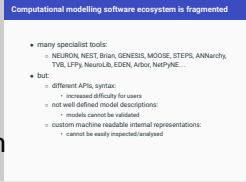
- many specialist tools:
  - NEURON, NEST, Brian, GENESIS, MOOSE, STEPS, ANNarchy, TVB, LFPy, NeuroLib, EDEN, Arbor, NetPyNE…
- but:
  - different APIs, syntax:
    - increased difficulty for users
  - not well defined model descriptions:
    - models cannot be validated



1. There are a lot of software tools out there for users to pick from, for different levels of modelling, optimisation, analysis.
2. For each stage.
3. But, they aren't designed to work together.
4. They have their own designs, their own APIs, syntax, model representation, and usually their own suite of custom utilities to work with their model representation.

Computational modelling software ecosystem is fragmented

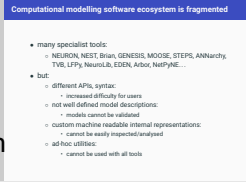
- many specialist tools:
  - NEURON, NEST, Brian, GENESIS, MOOSE, STEPS, ANNarchy, TVB, LFPy, NeuroLib, EDEN, Arbor, NetPyNE…
- but:
  - different APIs, syntax:
    - increased difficulty for users
  - not well defined model descriptions:
    - models cannot be validated
  - custom machine readable internal representations:
    - cannot be easily inspected/analysed



1. There are a lot of software tools out there for users to pick from, for different levels of modelling, optimisation, analysis.
2. For each stage.
3. But, they aren't designed to work together.
4. They have their own designs, their own APIs, syntax, model representation, and usually their own suite of custom utilities to work with their model representation.

Computational modelling software ecosystem is fragmented

- many specialist tools:
  - NEURON, NEST, Brian, GENESIS, MOOSE, STEPS, ANNarchy, TVB, LFPy, NeuroLib, EDEN, Arbor, NetPyNE…
- but:
  - different APIs, syntax:
    - increased difficulty for users
  - not well defined model descriptions:
    - models cannot be validated
  - custom machine readable internal representations:
    - cannot be easily inspected/analysed
  - ad-hoc utilities:
    - cannot be used with all tools



1. There are a lot of software tools out there for users to pick from, for different levels of modelling, optimisation, analysis.
2. For each stage.
3. But, they aren't designed to work together.
4. They have their own designs, their own APIs, syntax, model representation, and usually their own suite of custom utilities to work with their model representation.

Makes computational neuroscience models

less

FAIR

(Findable, Accessible, Interoperable, Reusable)

Makes computational neuroscience models
less
FAIR
(Findable, Accessible, Interoperable, Reusable)

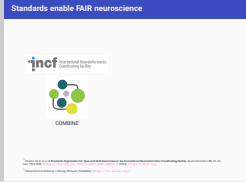
1. This means that for example, a model written in simulator A, say NEURON, cannot just be re-used in another simulator.
2. In fact, because a majority of these tools do not have a well defined model description, even re-using models developed in the same simulator can be quite hard.
3. It takes a lot of human resources to translate/convert models to be able to re-use them.
4. It also makes it very hard to study or analyse these models.

Standards enable FAIR neuroscience

COMBINE

[1] Abrams, M. B. *et al.* **A Standards Organization for Open and FAIR Neuroscience: the International Neuroinformatics Coordinating Facility.** *Neuroinformatics* **20,** 25–36. ISSN: 1559-0089. https://doi.org/10.1007/s12021-020-09509-0 (2022): https://incf.org

[1] COmputational Modeling in BIology NEtwork (COMBINE): https://co.mbine.org/



1. Now, this isn't a problem unique to computational neuroscience, or even neuroscience.
2. Multiple scientific fields have run into this issue, and the answer that they've all come up with is to standardise.
3. Standards allow the representation of data and models in specific, agreed formats.
4. Once a standard is agreed upon, everyone can target it—tools, representations, utilities.
5. If one knows what the data is going to look like, one can then develop tools and APIs around it.
6. And instead of everyone writing a tool for their own standard, every tool anyone writes for the one standard can be used with everyone's data.

Standards enable FAIR neuroscience

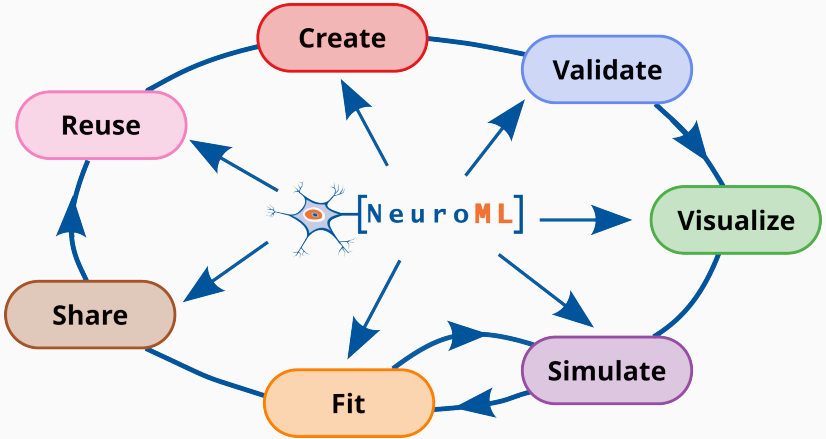
COMBINE

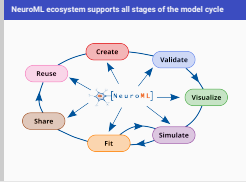
[1] Abrams, M. B. *et al.* **A Standards Organization for Open and FAIR Neuroscience: the International Neuroinformatics Coordinating Facility.** *Neuroinformatics* **20,** 25–36. ISSN: 1559-0089. https://doi.org/10.1007/s12021-020-09509-0 (2022): https://incf.org

[1] COmputational Modeling in BIology NEtwork (COMBINE): https://co.mbine.org/

1. Now, this isn't a problem unique to computational neuroscience, or even neuroscience.
2. Multiple scientific fields have run into this issue, and the answer that they've all come up with is to standardise.
3. Standards allow the representation of data and models in specific, agreed formats.
4. Once a standard is agreed upon, everyone can target it—tools, representations, utilities.
5. If one knows what the data is going to look like, one can then develop tools and APIs around it.
6. And instead of everyone writing a tool for their own standard, every tool anyone writes for the one standard can be used with everyone's data.

NeuroML ecosystem supports all stages of the model cycle





1. The idea being that by being the standard, various tools that support various stages of the model life cycle can then work together.

NeuroML ecosystem

- standard/specification
- software ecosystem

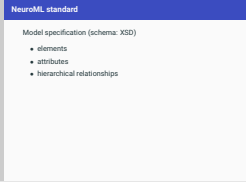
NeuroML ecosystem
- standard/specification
- software ecosystem

1. It consists of two components. The specification or the standard, and the software that adhere to this specification.

NeuroML standard

Model specification (schema: XSD)

- elements
- attributes
- hierarchical relationships



1. The standard itself has two different components.
2. There's the schema, which formally specifies the model description—what elements, attributes are valid, and how they related to each other.
3. The next is the LEMS description of the model—the dynamics. We call this the Component type declaration.

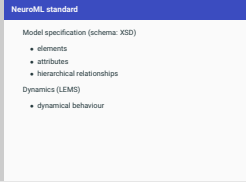
NeuroML standard

Model specification (schema: XSD)

- elements
- attributes
- hierarchical relationships

Dynamics (LEMS)

- dynamical behaviour



1. The standard itself has two different components.
2. There's the schema, which formally specifies the model description—what elements, attributes are valid, and how they related to each other.
3. The next is the LEMS description of the model—the dynamics. We call this the Component type declaration.

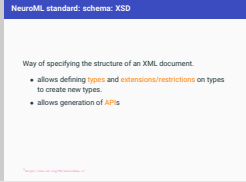
NeuroML standard: schema: XSD

Way of specifying the structure of an XML document.

- allows defining types and extensions/restrictions on types to create new types.
- allows generation of APIs

[1] https://www.w3.org/TR/xmlschema-1/

1. That's it. We'll see an example now.



NeuroML standard: schema: XSD

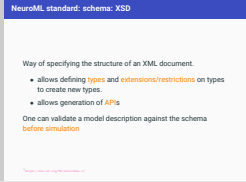
Way of specifying the structure of an XML document.

- allows defining types and extensions/restrictions on types to create new types.
- allows generation of APIs

One can validate a model description against the schema before simulation

[1] https://www.w3.org/TR/xmlschema-1/

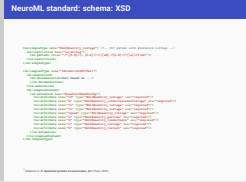
1. That's it. We'll see an example now.



NeuroML standard: schema: XSD

```xml
<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="Izhikevich2007Cell">
  <xs:annotation>
    <xs:documentation>Cell based on ...>
    </xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="BaseCellMembPotCap">
      <xs:attribute name="v0" type="Nml2Quantity_voltage" use="required"/>
      <xs:attribute name="k" type="Nml2Quantity_conductancePerVoltage" use="required"/>
      <xs:attribute name="vr" type="Nml2Quantity_voltage" use="required"/>
      <xs:attribute name="vt" type="Nml2Quantity_voltage" use="required"/>
      <xs:attribute name="vpeak" type="Nml2Quantity_voltage" use="required"/>
      <xs:attribute name="a" type="Nml2Quantity_pertime" use="required"/>
      <xs:attribute name="b" type="Nml2Quantity_conductance" use="required"/>
      <xs:attribute name="c" type="Nml2Quantity_voltage" use="required"/>
      <xs:attribute name="d" type="Nml2Quantity_current" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

[1] Izhikevich, E. M. ***Dynamical systems in neuroscience.*** (MIT Press, 2007)



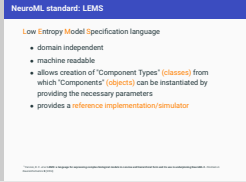
1. The schema is defined as an XSD: XML schema document that formally defines what an XML file can look like.
2. In this example, of a simple cell model, the Izhikevich model, this lists what its attributes are.
3. In programming jargon, we're defining the structure of the class—what parameters/attributes can an instance/object of class contain.

NeuroML standard: LEMS

Low Entropy Model Specification language

- domain independent
- machine readable
- allows creation of "Component Types" (classes) from which "Components" (objects) can be instantiated by providing the necessary parameters
- provides a reference implementation/simulator

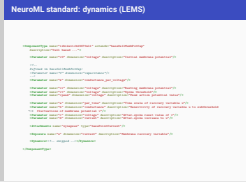
[1] Cannon, R. C. *et al.* **LEMS: a language for expressing complex biological models in concise and hierarchical form and its use in underpinning NeuroML 2.** *Frontiers in Neuroinformatics* **8** (2014)



1. The next is the LEMS description of the model—the dynamics. We call this the Component type declaration.

NeuroML standard: dynamics (LEMS)

```
<ComponentType name="izhikevich2007Cell" extends="baseCellMembPotCap"
    description="Cell based ...">

    <Parameter name="v0" dimension="voltage" description="Initial membrane potential"/>

    <!--
    Defined in baseCellMembPotCap:
    <Parameter name="C" dimension="capacitance"/>
    -->
    <Parameter name="k" dimension="conductance_per_voltage"/>

    <Parameter name="vr" dimension="voltage" description="Resting membrane potential"/>
    <Parameter name="vt" dimension="voltage" description="Spike threshold"/>
    <Parameter name="vpeak" dimension="voltage" description="Peak action potential value"/>

    <Parameter name="a" dimension="per_time" description="Time scale of recovery variable u"/>
    <Parameter name="b" dimension="conductance" description="Sensitivity of recovery variable u to subthreshold
    ↪  fluctuations of membrane potential v"/>
    <Parameter name="c" dimension="voltage" description="After-spike reset value of v"/>
    <Parameter name="d" dimension="current" description="After-spike increase to u"/>

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

    <Exposure name="u" dimension="current" description="Membrane recovery variable"/>

    <Dynamics><!-- snipped --></Dynamics>

</ComponentType>
```

1. Here is the LEMS component type definition, without the dynamics for the moment.
2. What you'll notice is that a lot of it is very similar to the XSD definition.

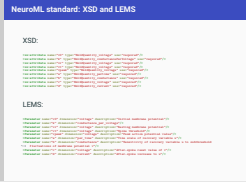
NeuroML standard: XSD and LEMS

XSD:

```
<xs:attribute name="v0" type="Nml2Quantity_voltage" use="required"/>
<xs:attribute name="k" type="Nml2Quantity_conductancePerVoltage" use="required"/>
<xs:attribute name="vr" type="Nml2Quantity_voltage" use="required"/>
<xs:attribute name="vt" type="Nml2Quantity_voltage" use="required"/>
<xs:attribute name="vpeak" type="Nml2Quantity_voltage" use="required"/>
<xs:attribute name="a" type="Nml2Quantity_pertime" use="required"/>
<xs:attribute name="b" type="Nml2Quantity_conductance" use="required"/>
<xs:attribute name="c" type="Nml2Quantity_voltage" use="required"/>
<xs:attribute name="d" type="Nml2Quantity_current" use="required"/>
```

LEMS:

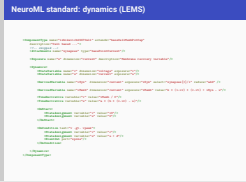
```
<Parameter name="v0" dimension="voltage" description="Initial membrane potential"/>
<Parameter name="k" dimension="conductance_per_voltage"/>
<Parameter name="vr" dimension="voltage" description="Resting membrane potential"/>
<Parameter name="vt" dimension="voltage" description="Spike threshold"/>
<Parameter name="vpeak" dimension="voltage" description="Peak action potential value"/>
<Parameter name="a" dimension="per_time" description="Time scale of recovery variable u"/>
<Parameter name="b" dimension="conductance" description="Sensitivity of recovery variable u to subthreshold
↪  fluctuations of membrane potential v"/>
<Parameter name="c" dimension="voltage" description="After-spike reset value of v"/>
<Parameter name="d" dimension="current" description="After-spike increase to u"/>
```

1. I've written the attributes/parameters together here so you can see that there's a one-to-one correspondence between the two model descriptions.

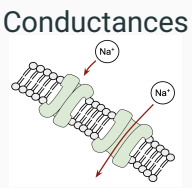
NeuroML standard: dynamics (LEMS)

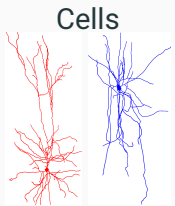
```
<ComponentType name="izhikevich2007Cell" extends="baseCellMembPotCap"
    description="Cell based ...">
    <!-- snipped -->
    <Attachments name="synapses" type="basePointCurrent"/>

    <Exposure name="u" dimension="current" description="Membrane recovery variable"/>

    <Dynamics>
        <StateVariable name="v" dimension="voltage" exposure="v"/>
        <StateVariable name="u" dimension="current" exposure="u"/>

        <DerivedVariable name="iSyn"  dimension="current" exposure="iSyn" select="synapses[*]/i" reduce="add" />

        <DerivedVariable name="iMemb" dimension="current" exposure="iMemb" value="k * (v-vr) * (v-vt) + iSyn - u"/>

        <TimeDerivative variable="v" value="iMemb / C"/>
        <TimeDerivative variable="u" value="a * (b * (v-vr) - u)"/>

        <OnStart>
            <StateAssignment variable="v" value="v0"/>
            <StateAssignment variable="u" value="0"/>
        </OnStart>

        <OnCondition test="v .gt. vpeak">
            <StateAssignment variable="v" value="c"/>
            <StateAssignment variable="u" value="u + d"/>
            <EventOut port="spike"/>
        </OnCondition>

    </Dynamics>
</ComponentType>
```

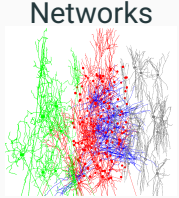



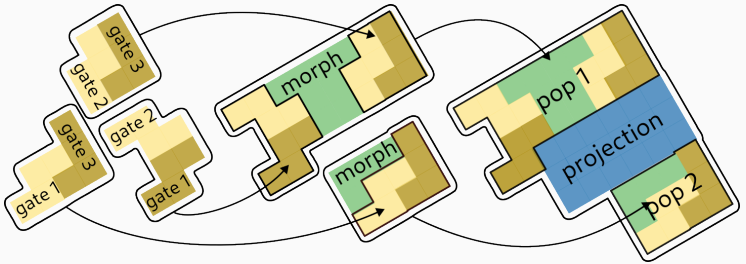
1. This now shows the dynamics of the component type.
2. It include information on how the states and time derivatives change, how the various variables interact.
3. This information is sufficient to then be able to create an object of this type and to simulate it.

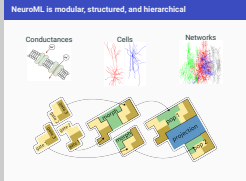
NeuroML is modular, structured, and hierarchical





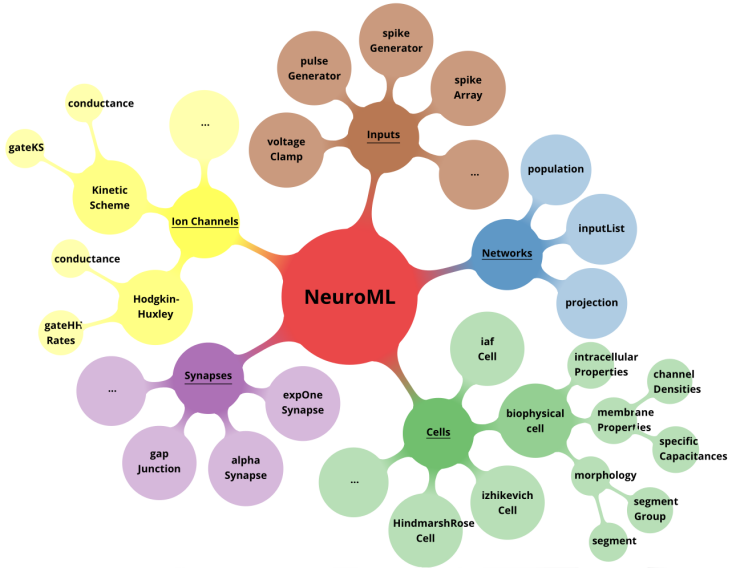




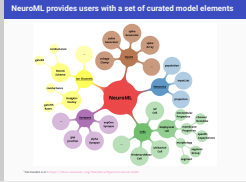


1. Hopefully this gives you an idea of how NeuroML is modular, structured, and hierarchical by design.
2. So, we like to think of it building blocks, and since the relationships between blocks is well defined, the blocks can be easily re-used and combined to create new ones.
3. So, there are three conductances here, for example—using the same three HH type gates.
4. These conductances can be used in different cells—here two cells with different morphologies.
5. The cells can then be used in populations to create a network, and so on.

NeuroML provides users with a set of curated model elements

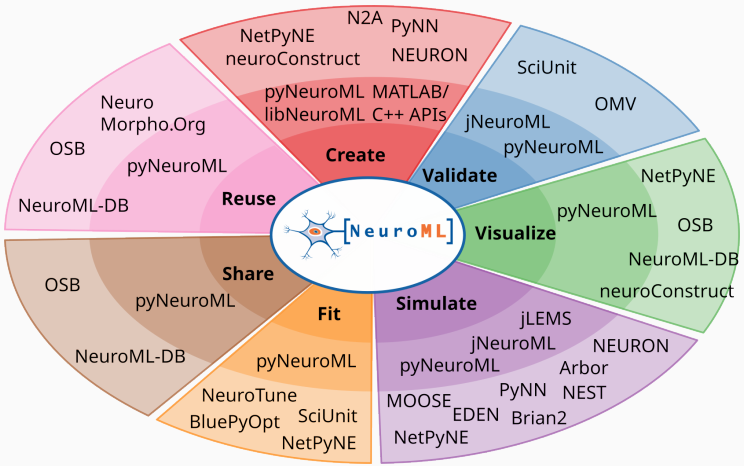


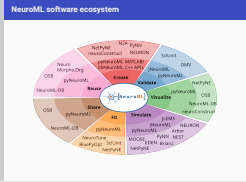
[1] Full standard is at: https://docs.neuroml.org/Userdocs/Specification.html



1. An important aspect of the standard is that it includes lots of commonly used model elements already.
2. So that users don't have to write these themselves, they can use the ones already provided.
3. The mind map shows you a sub-set of model elements included in the NeuroML standard.

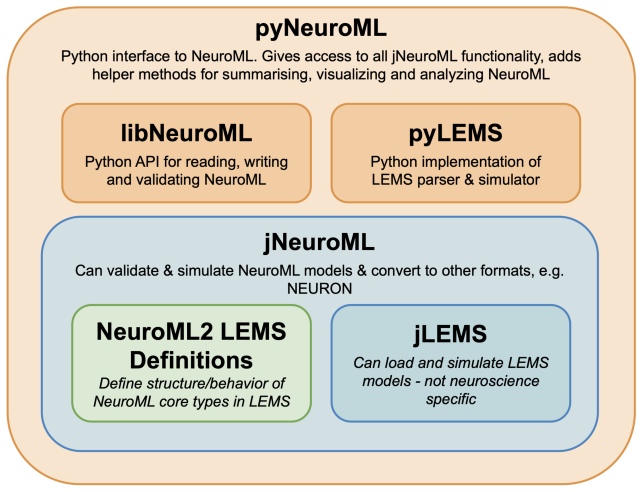
NeuroML software ecosystem

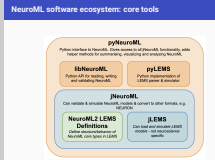




1. An important aspect of the standard is that it includes lots of commonly used model elements already.
2. So that users don't have to write these themselves, they can use the ones already provided.
3. The mind map shows you a sub-set of model elements included in the NeuroML standard.

NeuroML software ecosystem: core tools





1. These tools form the core NeuroML tools—ones that the NeuroML Editors, we develop and maintain
2. These provide the basic APIs required to work with NeuroML models—to create them, to read, write, and modify them
3. PyNeuroML is the suggested tool for use—we don't want people writing XML.

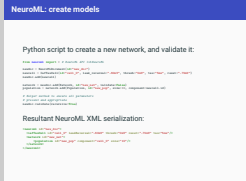
NeuroML: create models

Python script to create a new network, and validate it:

```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)
```

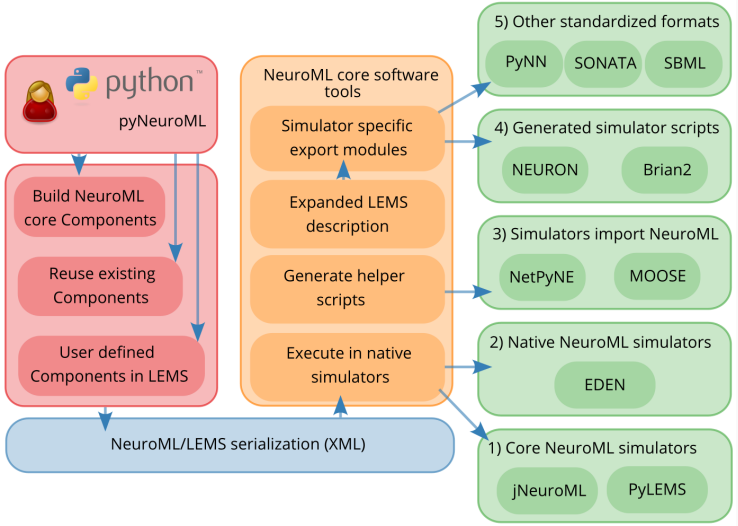
Resultant NeuroML XML serialization:

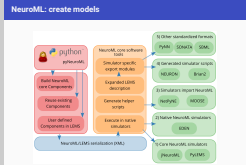
```xml
<neuroml id="new_doc">
  <iafTauCell id="cell_0" leakReversal="-60mV" thresh="0mV" reset="-70mV" tau="5ms"/>
  <network id="new_net">
      <population id="new_pop" component="cell_0" size="10"/>
  </network>
</neuroml>
```

1. Now, we do not want users to write XML at all
2. The Python API is generated from the schema, and one can use this to write models.

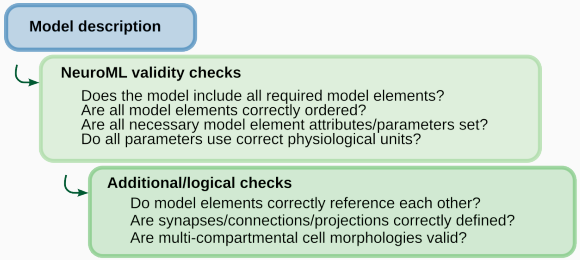
NeuroML: create models

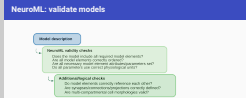




1. Once a model has been created, it is stored in its NeuroML/LEMS XML form
2. Because LEMS is formally defined and machine readable, this can now be easily converted into any other require format
3. The different simulators support NeuroML in different ways
4. For example, for NEURON, we generate NEURON scripts
5. But NetPyNE imports NeuroML to convert it into its own internal format
6. The advantage here is that simulator developers are free to choose how they want to support NeuroML

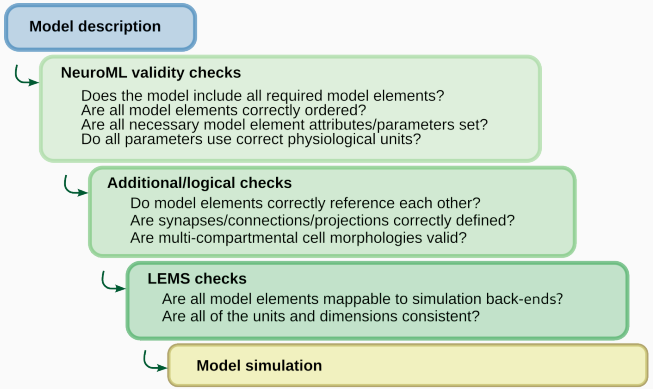
NeuroML: validate models

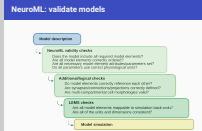




1. We've already touched on validation a little, but let's look at it in more detail
2. The first level of validation is against the schema—is the model valid, does it have the right structure
3. Since we have access to the complete model description, we can also run additional global checks—are connections correctly defined, for example?
4. Once the model has been converted to its expanded LEMS form, we run more checks
5. Does the model support the requested simulator?
6. LEMS is dimension aware—so it checks to see if units and dimensions are consistent
7. We can now simulate the model being fairly confident that it has been defined correctly

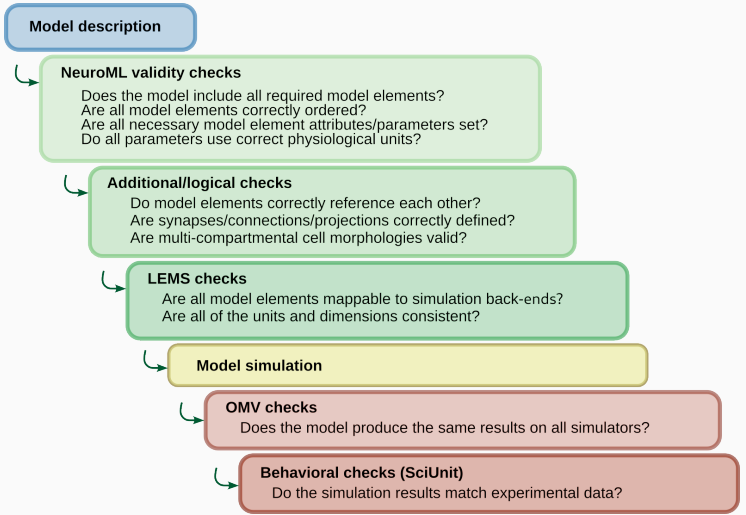
NeuroML: validate models

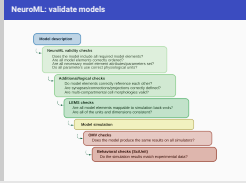




1. We've already touched on validation a little, but let's look at it in more detail
2. The first level of validation is against the schema—is the model valid, does it have the right structure
3. Since we have access to the complete model description, we can also run additional global checks—are connections correctly defined, for example?
4. Once the model has been converted to its expanded LEMS form, we run more checks
5. Does the model support the requested simulator?
6. LEMS is dimension aware—so it checks to see if units and dimensions are consistent
7. We can now simulate the model being fairly confident that it has been defined correctly

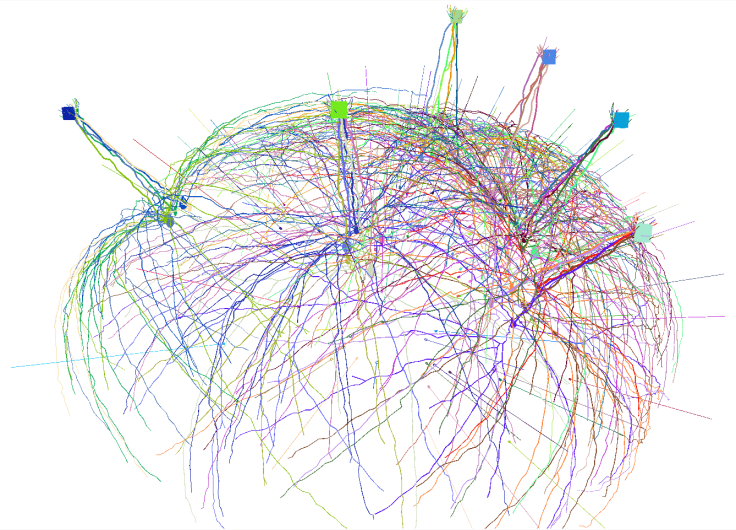
NeuroML: validate models



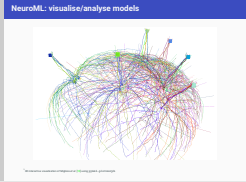


1. We've already touched on validation a little, but let's look at it in more detail
2. The first level of validation is against the schema—is the model valid, does it have the right structure
3. Since we have access to the complete model description, we can also run additional global checks—are connections correctly defined, for example?
4. Once the model has been converted to its expanded LEMS form, we run more checks
5. Does the model support the requested simulator?
6. LEMS is dimension aware—so it checks to see if units and dimensions are consistent
7. We can now simulate the model being fairly confident that it has been defined correctly

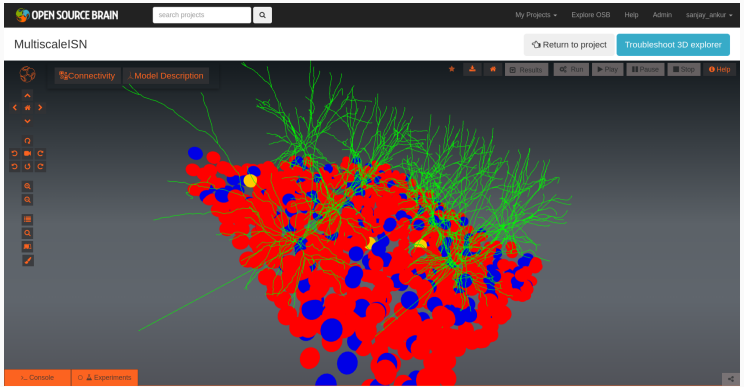
NeuroML: visualise/analyse models



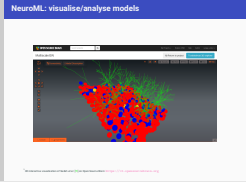
[1] 3D interactive visualisation of Migliore *et al.* [10] using `pynml-plotmorph`



NeuroML: visualise/analyse models

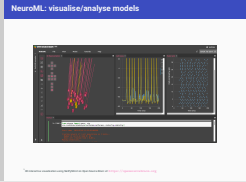


[1] 3D interactive visualisation of Sadeh *et al.* [11] on Open Source Brain: https://v1.opensourcebrain.org

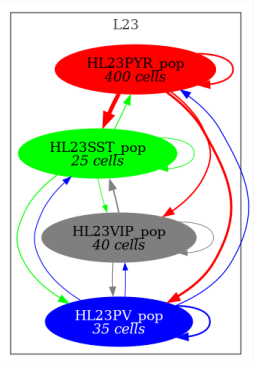


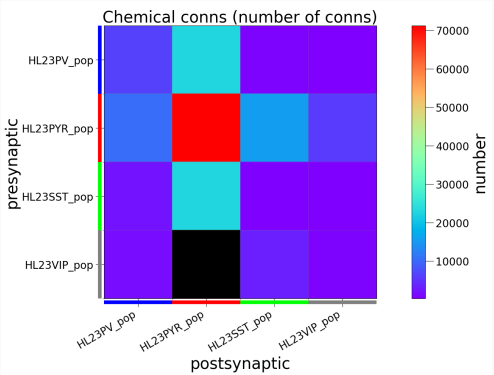
NeuroML: visualise/analyse models

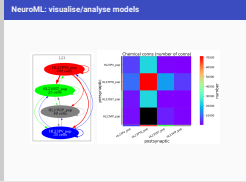
[1] 3D interactive visualisation using NetPyNE-UI on Open Source Brain v2: https://opensourcebrain.org



NeuroML: visualise/analyse models







NeuroML: simulate models

- Example simulation: neuron/netpyne

NeuroML: simulate models

- Example simulation: neuron/netpyne

NeuroML: fit models

- Figure from docs
- Mention inspyred

NeuroML: fit models
- Figure from docs
- Mention inspyred

NeuroML: share and re-use models

- GitHub, OSBv1, OSBv2, NeuroML-DB

NeuroML: share and re-use models
- GitHub, OSBv1, OSBv2, NeuroML-DB

NeuroML: Documentation

- Jupyterbook

NeuroML: Documentation

- Jupyterbook

NeuroML: projects

- GSoC, Outreachy, good computer science students

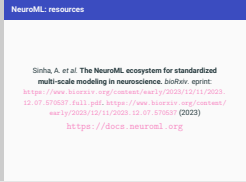
NeuroML: projects

- GSoC, Outreachy, good computer science students

NeuroML: resources

Sinha, A. *et al.* **The NeuroML ecosystem for standardized multi-scale modeling in neuroscience.** *bioRxiv.* eprint: https://www.biorxiv.org/content/early/2023/12/11/2023.12.07.570537.full.pdf. https://www.biorxiv.org/content/early/2023/12/11/2023.12.07.570537 (2023)

https://docs.neuroml.org



NeuroML: the APIs

- Python API

NeuroML: the APIs
- Python API