NeuroArch: A Graph dB for Querying and Executing Fruit Fly Brain Circuits

Neurokernel RFC #5 v1.0

Lev E. Givon, Aurel A. Lazar, Nikul H. Ukani
Bionet Group
Department of Electrical Engineering
Columbia University, New York, NY 10027

December 31, 2015

Abstract

NeuroArch is a database for codifying knowledge about fruit fly brain circuits. It is designed with two user communities in mind: (i) neurobiologists interested in querying the database to address questions regarding neuroanatomy, neural circuits, neurons, synapses, neurotransmitters, and gene expression, and (ii) computational neuroscientists interested in the instantiation of models of neural circuits and architectures, their program execution, and validation of hypotheses regarding brain function. A key aim of NeuroArch is to provide an interface between the concerns of these two communities. To this end, NeuroArch defines a data model for representation of both biological data and model structure and the relationships between them within a single graph database. When coupled with a powerful interface for querying both types of data within the database in a uniform manner, this representation enables neurobiologists to benefit from high-level organization of fruit fly data, while model designers can capitalize upon the integration of biological data from multiple sources. This document describes the requirements and design of NeuroArch, and details how it can be used with the Neurokernel framework to accelerate collaborative development of models of the fruit fly brain. A brief discussion of future development plans is also provided.

 DOI:10.5281/zenodo.44225

Contents

 DOI:10.5281/zenodo.44225

 DOI:10.5281/zenodo.44225

1 Introduction

The Neurokernel framework enables collaboration between multiple researchers to accelerate the development of fly brain models [9]. However, it lacks several elements essential to the efficient translation of biological knowledge into fruit fly circuit models and their subsequent refinement based upon model execution results.

Construction of neuropil models in the Neurokernel environment currently requires either manual implementation of the models in Python or specification of the graph of neuron and synapse model instances along with their required parameter values in a supported format such as the Graph Exchange Format (GEXF) [1]. The ad hoc construction of models from biological data complicates and hampers revision of models in light of either execution results or new experimental knowledge. Moreover, the various available fruit fly datasets employ multiple storage formats that cannot always be easily queried. These include but are not limited to data relating to the connectome, morphology, genetics, simulation specification, and physiology. The lack of common representation of domain knowledge hinders interaction and collaboration which ultimately slows the progress in the field.

NeuroArch is a software package that addresses the above limitations by providing a database for specification and storage of both biological data regarding the fruit fly brain and executable models built upon that data within a single graph database. NeuroArch employs a data model that preserves the structural and semantic relationships between different biological and modeling objects. Its query interface provides an object-graph mapping (OGM) that exploits this data model to enable both neurobiologists and neural circuit model designers to easily perform sophisticated queries relevant to their respective needs without having to explicitly specify complex query strings. NeuroArch aims to not only be useful to both neurobiologists and neural circuit modelers individually, but also serve as an interface between them.

The explosion in available experimental neuroscience data and increasing number of neural circuit models developed in recent years has motivated the development of a range of data repositories and neuroinformatic tools for publicly sharing neurobiological and modeling data with the research community. These range from downloadable connectome datasets [2, 3] to platforms designed to clearly annotate and classify biological structures in the fly brain, aid directed and intelligent online access to anatomical and genetic data, integrate and synthesize data from multiple sources, and enable new and enhanced analyses of available data [1, 4, 5, 6, 7, 8, 18, 23]. These are paralleled by databases and related technologies for specifying and disseminating

[1] http://www.gexf.net

 DOI:10.5281/zenodo.44225

neural models [10, 11, 12, 14]. NeuroArch aspires to prosecute similar goals as these projects, but within the more focused context of combining biological data, metadata annotations, and model structures into linked ontologies specific to a single organism that can enable both neurobiological inquiries spanning different ontologies and construction of more accurate and comprehensive brain models informed by multiple data sources.

NeuroArch also bears similarity to Bio4j, an open-source platform for integration of open bioinformatic datasets using typed graph models [19]. Like NeuroArch, Bio4j aims to link biological data (e.g., protein sequences) with semantic data (e.g., protein functional annotations, gene ontologies, organism taxonomies, enzyme nomenclature) from multiple sources within a single graph database to enable reasoning based upon the structure of the data in addition to the individual data points. Bio4j provides a data model that addresses how elements from different data sources are connected in order to obtain conclusions not achievable using a single unintegrated source; it also provides a Domain Specific Language (DSL) to facilitate creation of the complex database queries required to navigate the various types of graph elements.

The remainder of this RFC is organized as follows: in § 2, we describe NeuroArch's high level requirements for data representation. We propose a data model based upon these requirements in § 3, and describe the mapping of this model into a graph database in § 4. In § 5, we discuss features of NeuroArch's API that exploit the data model to fulfill some of the requirements described in § 2, and present demonstrations of the API's functionality in § 6. Finally, we provide a summary and discuss plans for future development of NeuroArch in § 7.

2 Data Representation Requirements

2.1 Information to be Represented

NeuroArch's database must be able to store data regarding both neurobiological circuits and the design of executable neural circuits that model their biological counterparts. The former includes data such as neuron and synapse structure and characteristics from sources such as EM reconstruction, transgenic lines and genetic data; the latter includes parameters of constituent component models and abstractions that describe a neural circuit's architecture. Since data regarding the same biological entities may be provided by different experimental data sources, NeuroArch must support concurrent representation of biological data with different origins to enable the incompleteness of data from one source to be complemented by data from a different source. Similarly, NeuroArch must support concurrent representation of multiple

 DOI:10.5281/zenodo.44225

versions of a single circuit designed by different parties or containing different design variations.

2.1.1 Biological Circuit Entities

Entities corresponding to biological data NeuroArch must support are listed below. Some of these entities correspond to sets or subdivisions of other biological entities, while others correspond to attributes of other entities.

Arborization Data Data regarding the arborization of dendrites within specific brain regions, e.g., the identity of the neurons that arborize within a specific glomerulus in the protocerebral bridge and the polarity of their respective neurites. This geometric data may be less detailed than neuron morphology data.

Biological Sensor A set of sensory neurons such as photoreceptors, olfactory sensory neurons, or mechanosensory cells.

Chemical Synapse A neurotransmitter-mediated connection between two neurons. Henceforth referred to as a synapse in the remainder of this RFC.

Data Source The source (e.g., a lab or research group) of a set of biological fly brain data.

Gap Junction A non-chemical connection between two neurons.

Genetic Data Data regarding the genetic line associated with other biological entities such as neurons or synapses.

Neural Circuit Motif A brain circuit other than (and typically smaller than) a neuropil, e.g., cartridge (in lamina), column (in medulla), channel (in antennal lobe), etc.

Neuron Morphology Data Data describing a neuron's geometry.

Neuron A single neuron, e.g., Tm-1, L1, etc.

Neuropil Any named anatomical region of the fly brain, e.g., lamina, medulla, etc. [15].

Neurotransmitter Associated with a specific neuron or synapse, e.g., histamine, acetylcholine, GABA, etc.

Species The species associated with a given set of biological data, e.g., *D. melanogaster*, *D. simulans*, *D. busckii*, etc.

Tracts A bundle of neuron axons at the mesoscopic scale, i.e., information regarding the individual neurons in the bundle may be absent even if knowledge regarding the endpoints and total number of axons is known.

2.1.2 Executable Circuit Entities

Entities required to represent executable circuit designs are listed below. Some of these entities represent architectural abstractions, while others (such as model parameters) correspond to attributes of other entities.

Axon Model An instance of a model of a neuron's axon.

Axon Hillock Model An instance of a model of a neuron's axon hillock, e.g., Leaky Integrate-and-Fire, Hodgkin-Huxley, Morris-Lecar (configured to emit spikes), etc.

Circuit Motif Model An instance of a neural circuit model, e.g., canonical circuits, composition rules in the fly vision system [16].

Communication Port A single input or output channel of an LPU model or pattern.

Dendrite Model An instance of a model of a neuron's dendrites.

Gap Junction Model An instance of a model of a gap junction between two neurons.

 DOI:10.5281/zenodo.44225

Inter-LPU Connectivity Pattern An instance of the connectivity between the ports exposed by two LPUs' interfaces.

LPU or Pattern Interface A set of ports exposed by an LPU or pattern for communication with those in the interfaces of other LPUs or patterns.

LPU An instance of a model of a specific neuropil that owns the objects that describe its internal design.

Membrane Model An instance of a model describing a neuron's membrane voltage, e.g., Morris-Lecar configured to not emit spikes.

Model Parameters Parameters associated with a functional model of structures such as a neuron, synapse, or gap junction.

Model Version An identifier distinguishing one version of an LPU or inter-LPU connectivity pattern from other instances of the same LPU or pattern.

Neuron Model An instance of a model of an entire neuron. This entity owns other entities that correspond to models of specific components of a neuron.

Synapse Model An instance of a model of a chemical synapse between two neurons.

2.2 Biological Circuit Query Requirements

1. The database should be able to store information from a variety of sources, e.g., EM reconstruction, transgenic lines, genetic data, etc. It should be possible to retrieve the data associated with a specific biological object that originates in different data sources.

2. NeuroArch must support querying of all stored biological data. For example, a neurobiologist should be able to retrieve all neurons associated with a particular genetic line whose neurotransmitter profile differs from that of the corresponding neurons in the wild type fruit fly.

 DOI:10.5281/zenodo.44225

3. Queries should be expressible in a high-level and intuitive fashion that enables neurobiologists to access high-level subdivisions (e.g., cartridges and columns in the vision neuropils) as well as lower level components such as neurons and synapses.

4. Queries should be able to incorporate and handle 'fuzzy' information. For example, it should be possible to represent data regarding a population of neurons with a characteristic associated with some fraction of the population rather than with individually identified neurons.

5. Queries should support names of biological structures and their synonyms as defined in existing anatomical ontologies [7].

6. NeuroArch should support representation of the confidence level of a dataset. For example, the confidence associated with synaptic connections inferred from overlapping arborizations should be assigned a lower level of confidence than that of connections obtained from EM reconstruction.

7. Data from multiple sources should be integrated such that queries can seamlessly traverse multiple sources even if the sources overlap or one dataset lacks information present in another dataset, e.g., one should be able to query neurons in multiple connected neuropils even if the data for those neuropils originates in different datasets.

2.3 Executable Circuits Query Requirements

1. NeuroArch must support defining and manipulating models whose respective internal structures may employ labeling schemes that potentially contain a greater or lesser number of abstraction levels than other models.

2. It should be possible to use biological information stored in NeuroArch to generate or update information of executable models of LPUs.

3. Queries in NeuroArch should be able to span all levels of model abstraction and access biological as well as modeling data. For example, it should be possible to retrieve the neurotransmitter profiles of the synapse model instances comprised by an LPU.

4. Data stored in NeuroArch should be accessible and/or modifiable in multiple modes suitable for different applications, i.e., as a subgraph (to preserve graph

relationships amongst components in the query results) or a table (to facilitate tabular or relational manipulations of the query results).

5. To enable circuit model execution, model objects defined in NeuroArch must correspond to code in Neurokernel's draft LPU implementation that numerically realize those models.

3 Data Model

NeuroArch's data model distinguishes between the representation of biological circuit data and executable circuit data. It employs two interconnected hierarchies to represent the information described in § 2.1. These hierarchies describe how entities at one level of granularity/abstraction are defined in terms of entities at some lower level of granularity/abstraction.

3.1 Biological Circuit Data and Its Subdivisions

Biological data in NeuroArch may be described at multiple levels of structural subdivisions that partition the data into subsets of increasingly finer granularity (Tab. 1). Subdivisions unique to specific neuropils (e.g., `Cartridge`, `Column`, `Channel`, etc.) may also be defined by the data model. Some of the information described in § 2.1.1 is deemed to be attributes of specific entities in the data model and therefore does not appear in Tab. 1.

Level	Name	Contains
Highest	`DataSource`	`GapJunction`, `Neuron`, `Synapse`
	`Species`	`Neuropil`
⋮	`BioSensor`	`Circuit`
	`Neuropil`	`Circuit`
	`Tract`	`Circuit`
⋮	`Circuit`	`GapJunction`, `Neuron`, `Synapse`
Lowest	`GapJunction`	
	`Neuron`	
	`Synapse`	

Table 1: Containment relationships between biological circuit entities in NeuroArch's data model.

 DOI:10.5281/zenodo.44225

3.2 Naming Scheme for Biological Data

Every biological entity defined in the fruit fly brain (some of which may correspond to sets of other entities) must be assigned a unique name. The naming scheme should in principle be extensible to other model organisms, e.g., *C. elegans*, zebra fish, mouse, etc. Unique names should include identifying information about the successive levels of subdivision associated with the entity in question (§ 2.1.1); this can be done employing a naming syntax analogous to that employed in Uniform Resource Identifiers (URIs) that exploits the hierarchy of subdivisions described in § 3.1:

```
/Species/BioSensor/Circuit/Neuron
/Species/Neuropil/Circuit/Neuron
/Species/Tract
```

Synapse and gap junction names should be based upon names of the neurons they connect, e.g., `Dm2_C3` could denote a synapse between presynaptic neuron `Dm2` and postsynaptic neuron `C3`. Synapse and gap junction names must be able to distinguish between multiple synapses or gap junctions between two neurons. For example, the following identifiers could be assigned to synapses between `R1` photoreceptors in a specific cartridge of the retina and `L1` neurons in the corresponding cartridge of the lamina. Note that the synapse is deemed to belong to the postsynaptic neuropil, i.e., the lamina, rather than the retina.

```
/Drosophila_melanogaster/Lamina/Cartridge0/R1_L1/0
/Drosophila_melanogaster/Lamina/Cartridge0/R1_L1/1
```

3.3 Data and Abstractions for Executable Circuits

Data in NeuroArch that represents elements of executable circuits may be described at multiple levels of structural abstraction (Tab. 2). As with biological data, additional objects and levels may be defined depending upon the structure of the LPU model:

Level	Name	Owns
Highest	`Species`	`LPU`, `Pattern`
⋮	`LPU`	`CircuitModel`
	`Pattern`	`Interface`
⋮	`Interface`	`Port`
	`CircuitModel`	`GapJunctionModel`, `NeuronModel`, `SynapseModel`
	`NeuronModel`	`AxonHillockModel`, `AxonModel`, `DendriteModel`, `MembraneModel`
Lowest	`AxonHillockModel`	
	`AxonModel`	
	`DendriteModel`	
	`GapJunctionModel`	
	`MembraneModel`	
	`SynapseModel`	

Table 2: Ownership relationships between executable circuit entities in NeuroArch's data model.

3.4 Combined Hierarchy of Biological and Executable Circuit Entities

Fig. 1 depicts the combined hierarchies of biological and executable circuit entities supported by the data model. NeuroArch permits additional entities beyond those described in Fig. 1 to be specified, provided that entities on all levels consistently employ ownership relationships. This permits storage of executable circuit models and biological datasets with differing levels of abstraction or structural detail.

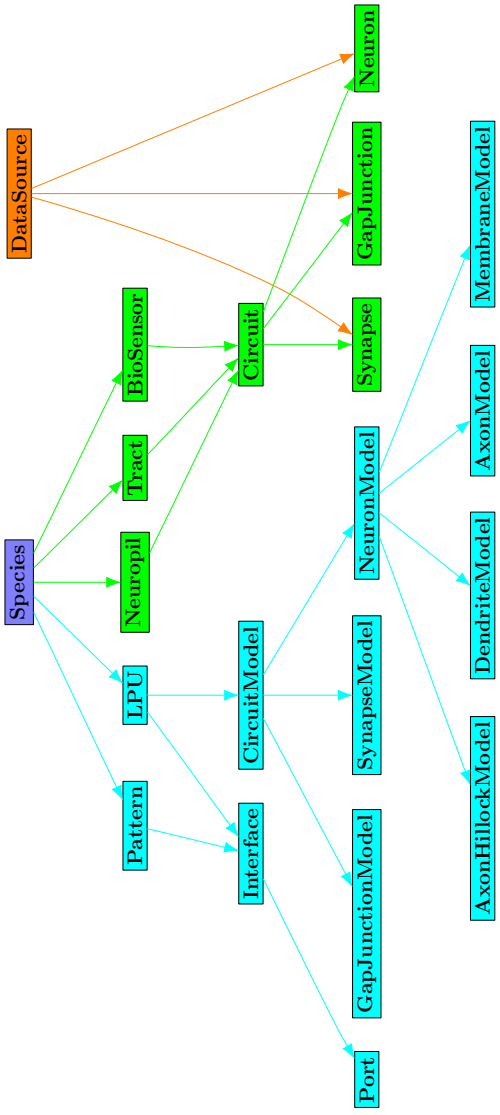


Figure 1: Objects and relationships in NeuroArch's data model. Green nodes denote biological circuit entities, while cyan nodes correspond to executable circuit entities; the purple node representing the species associated with specific biological or executable circuit entities is the root of both hierarchies. Green edges denote containment of lower level biological circuit entities in higher level subdivisions of the fly brain. Cyan edges denote ownership of executable circuit entities at lower levels of abstraction by those at higher levels of abstraction. Nodes representing actual neurons and synapses contained by a data source are connected to the orange data source node by orange edges representing containment.

 DOI:10.5281/zenodo.44225

4 Mapping the Data Model into an Object Graph Database

NeuroArch is implemented in Python and built upon the open-source database OrientDB[2]. This backend choice was made because of OrientDB's multi-model architecture that combines graph database support with NOSQL document storage features, its support for both a built-in SQL-like query language and the Gremlin[3] graph traversal language supported by many graph databases, and the availability of an actively developed Python interface[4] to the database. OrientDB also permits definition of node and edge types that subclass existing node and edge types; NeuroArch exploits this feature to enable the extension of the data model to include new node types required to represent biological structures or executable circuit elements not defined in Tab. 1 or 2.

4.1 Supported Relationships

Relationships between nodes in NeuroArch's database may either represent containment or ownership of one node by another (in the sense that one node represents a physical subdivision or lower level of abstraction than the node that contains or owns it) or the transmission of information between nodes. These relationships are depicted in Figs. 2 and 3, respectively.

[2] http://orientdb.com
[3] http://github.com/tinkerpop/gremlin/
[4] http://github.com/ostico/pyorient

 DOI:10.5281/zenodo.44225

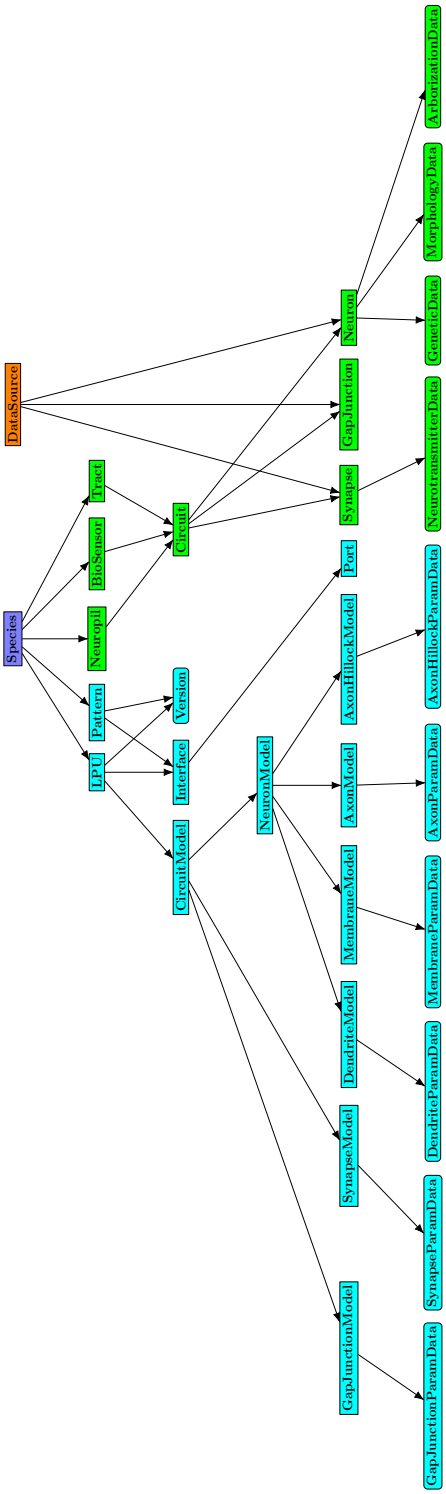


Figure 2: Object types and containment/ownership relationships in NeuroArch's database. Green nodes denote objects corresponding to biological circuit entities, while cyan nodes denote objects corresponding to executable circuit entities. Rectangular nodes correspond to entities defined by NeuroArch's data model that are mapped directly to database object types, while rounded nodes correspond to additional database object types that contain attributes of certain entities in the data model. Black edges represent both containment of lower level biological circuit objects by objects corresponding to higher level subdivisions and ownership of lower level executable circuit objects by objects corresponding to higher level abstractions.

 DOI:10.5281/zenodo.44225

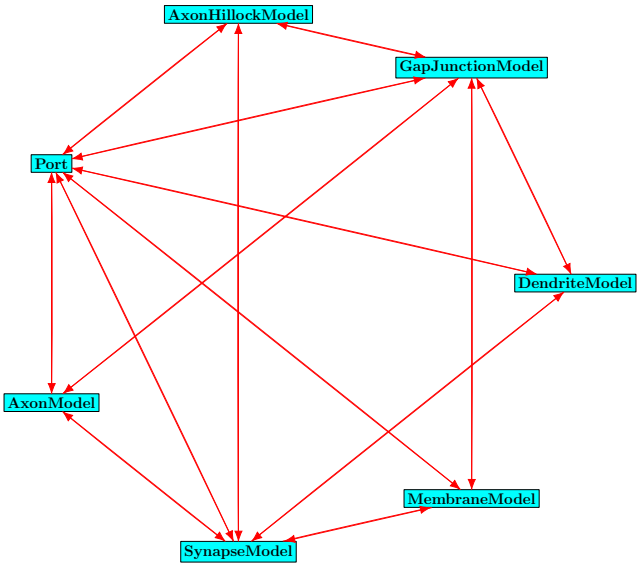


Figure 3: Data transmission relationships (red) between executable circuit objects (cyan) in NeuroArch's database.

4.2 Storage of Biological Data Objects

Most of the objects in NeuroArch's data model can be mapped directly into nodes in a graph database. In order to facilitate certain queries, data attributes associated with specific objects are mapped to additional nodes in the database that are linked to those that represent the objects that own them. For example, a `Neuron` object may own various descriptive data such as anatomical or genetic information; these data are stored in `MorphologyData`, `ArborizationData`, and `GeneticData` nodes respectively (Tab. 3).

Name	Owned by
`NeurotransmitterData`	`Synapse`
`MorphologyData`	`Neuron`
`GeneticData`	`Neuron`
`ArborizationData`	`Neuron`

Table 3: Objects used to store biological data.

4.3 Storage of Executable Circuit Data Objects

As with the biological data objects described in § 4.2, attributes of objects representing components of executable circuits may be mapped to separate nodes to facilitate certain queries (Tab. 4).

Name	Owned by
`AxonParamData`	`AxonModel`
`AxonHillockParamData`	`AxonHillockModel`
`DendriteParamData`	`DendriteModel`
`GapJunctionParamData`	`GapJunctionModel`
`MembraneParamData`	`MembraneModel`
`SynapseParamData`	`SynapseModel`

Table 4: Objects used to store executable circuit component data.

4.4 Naming and Storage of Multiple Model Versions

To enable the evaluation of different instances of a single neural circuit, NeuroArch must support storage of multiple versions of each LPU and inter-LPU connectivity pattern. Different versions of a single LPU or pattern are distinguished in NeuroArch's database by attaching a `Version` node containing a unique identifier to each node that respectively describe a particular version of the `LPU` or `Pattern` circuit in question (Fig. 4). Each `LPU` or `Pattern` node instance owns its own fully independent copy of the subgraph of lower level components that describe its version.

4.4.1 Relating Biological Data to Modeling Data

While the process of developing models of neural circuits in the fly brain requires support for simultaneous storage of multiple versions of a single LPU, biological data loaded from a particular data source is expected to remain static. NeuroArch therefore must store only one copy of each biological dataset to avoid redundancy. Each data source must be clearly identified in NeuroArch's database (§ 2.1.1, Fig. 2).

4.5 An Example - Representation of the Lamina and Retina

As an example of how NeuroArch's data model may be used and extended to represent specific regions in fruit fly brain, structures within the *Drosophila* lamina and

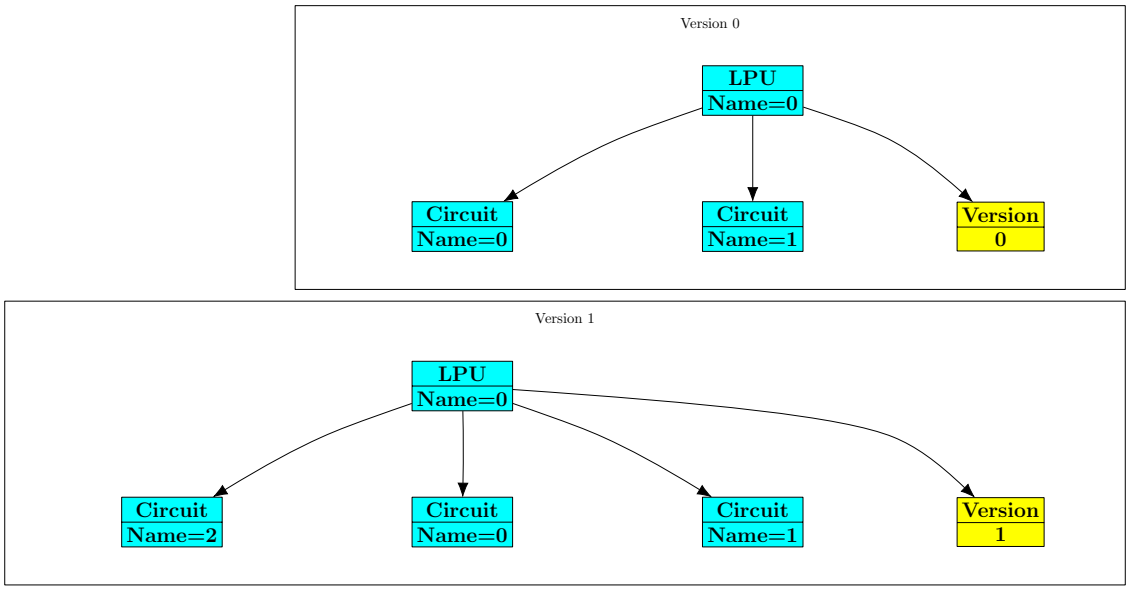
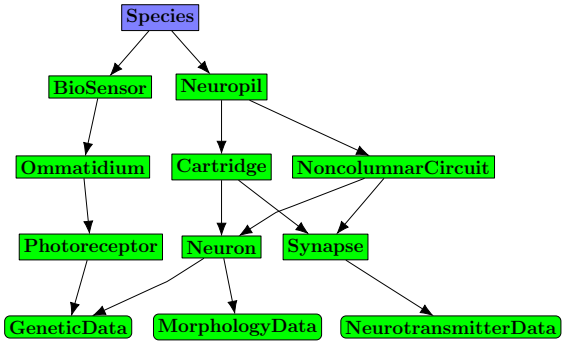
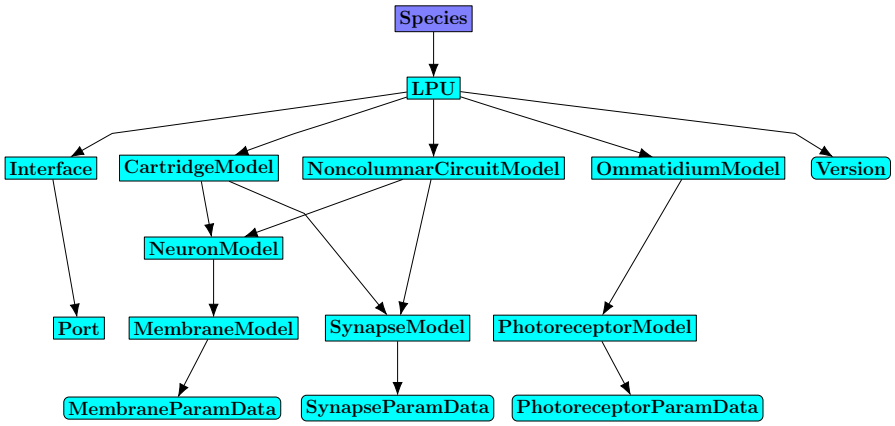


Figure 4: Representation of multiple versions of a single `LPU` (with name 0) using an additional `Version` node (yellow). Version 1 of the sample LPU differs from version 0 by virtue of the presence of an additional `Circuit` node in its subgraph.

retina as described in [16] can be mapped to the data model as depicted in Figs. 5 and 6 and Tab. 5.



(a) Containment relationships between biological circuit objects in the lamina and retina. Biological circuit node types specific to the lamina and retina descended from those in Fig. 2 are listed in Tab. 5a.



(b) Ownership relationships between executable circuit objects in the lamina and retina. Executable circuit node types specific to the lamina and retina descended from those in Fig. 2 are listed in Tab. 5b.

Figure 5: Containment/ownership relationships between biological and executable circuit database objects required to represent the lamina and retina. Rounded nodes represent attributes of entities in NeuroArch's data model that are mapped to nodes in NeuroArch's database (see Fig. 2)

Node Type	Parent Type	Instance Name Examples
BioSensor		Retina
Cartridge	Circuit	Cart1..Cart768
Neuron		L1..L6, Am, Lawf, C2, C3, T1
Neuropil		Lamina
NoncolumnarCircuit	Circuit	AmacrineCircuit
Ommatidium	Circuit	Cart1..Cart768
Photoreceptor	Neuron	R1..R6
Species		D. Melanogaster
Synapse		R1_L1, etc.

(a) Node types required to represent biological circuit entities in the lamina and retina.

Node Type	Parent Type	Instance Name Examples
CartridgeModel	CircuitModel	Cart1..Cart768
Interface		Lamina, Retina
LPU		Lamina, Retina
MembraneModel		L1..L6, Am, Lawf, C2, C3, T1
MembraneParamData		V1..V4, phi, etc.
NoncolumnarModel	CircuitModel	AmacrineCircuit
OmmatidiumModel	CircuitModel	Cart1..Cart768
Port		/lam/gpot/out0, etc.
PhotoreceptorModel	NeuronModel	R1..R6
PhotoreceptorParamData	MembraneParamData	R1..R6
SynapseModel		R1_L1, etc.
SynapseParamData		power, delay, etc.
Version		0, my_lpu, etc.

(b) Node types required to represent executable circuit entities in the lamina and retina.

Table 5: Node types required to represent the lamina and retina in NeuroArch's database. The sample names for instances of these nodes are illustrative; other names may be used as appropriate.

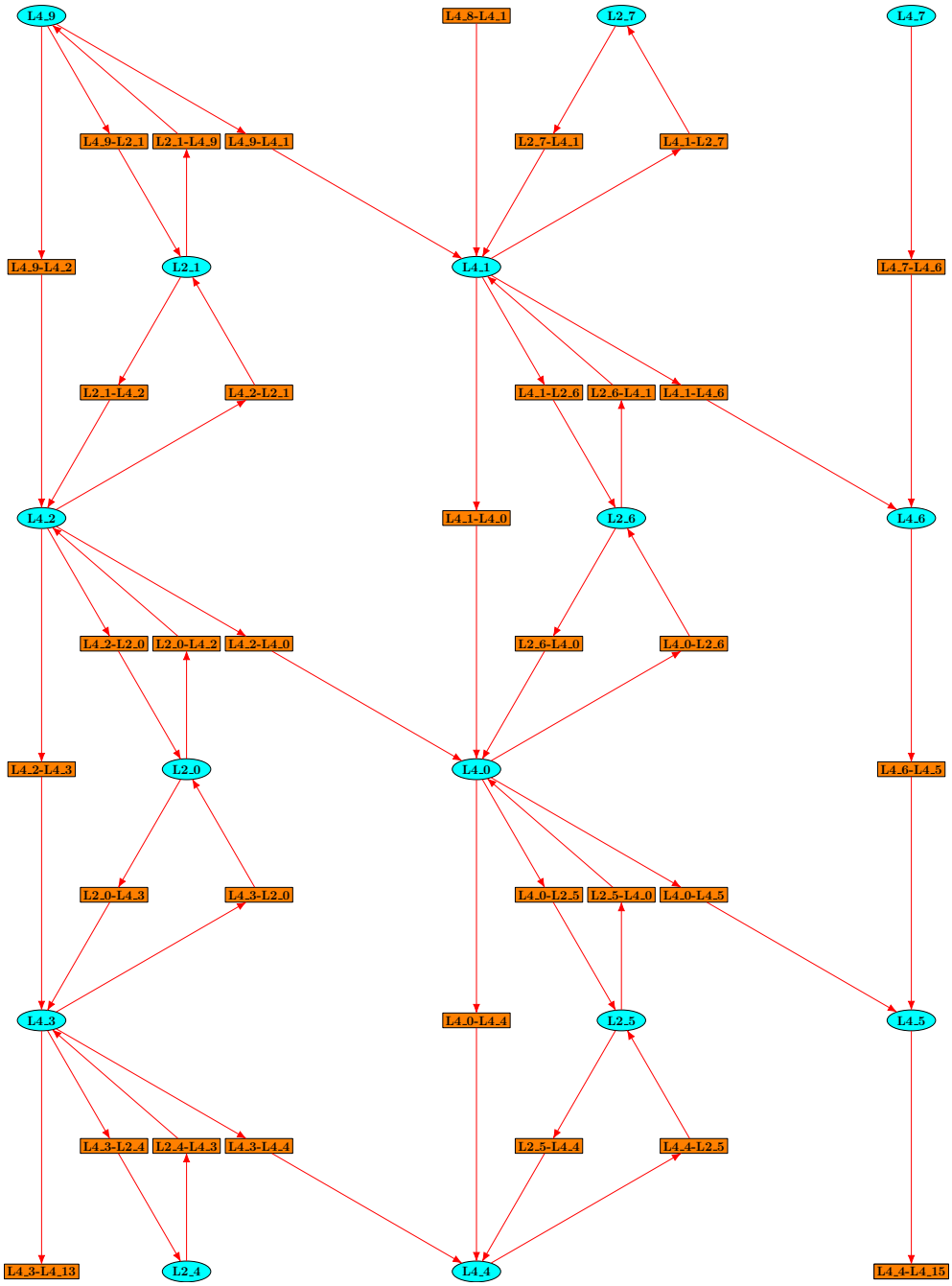


Figure 6: Data transmission (red) relationships between a subset of the circuit design components of the lamina LPU. This diagram only depicts the nodes corresponding to L2 and L4 neurons in several adjacent cartridges (cyan) and synapses between them (orange).

5 NeuroArch Application Programming Interface

Although the OrientDB graph database employed by NeuroArch supports powerful graph queries via its dialect of SQL and the Gremlin graph traversal language [22], the complexity of such queries can rapidly increase depending on the number of different elements in the database and the nature of the traversal that must be performed to obtain the query results. To obviate the need to explicitly construct such queries, NeuroArch provides a programming interface for generating useful queries against stored data that does not require explicit specification of a complex low-level query string.

5.1 Object Graph Mapping

NeuroArch exposes model data via an object graph mapping (OGM) that not only encapsulates individually stored elements, but also enables one to perform a selection of complex queries without having to express them in OrientDB SQL or Gremlin. The OGM provides methods associated with each object that dynamically construct and execute queries. This approach is similar to the concept of object relational mapping (ORM) used to interface with data models stored in relational databases. Two key differences between NeuroArch's OGM and that of currently available general-purpose OGMs are (i) its use of the hierarchical data model described in § 3 to enable extraction of subcircuits owned by nodes corresponding to specific subdivisions of biological components or circuit abstractions; (ii) the ability to use the subgraph extracted by an OGM query as the starting point for traversals by subsequent queries or as an operand that may be passed to graph operators (§ 5.3). To enable subsequent reuse of query results by subsequent queries or graph operations, NeuroArch permits optional storage of an extracted subgraph in its graph database. This subgraph can be discarded when no longer needed.

5.2 Supported Queries

NeuroArch's OGM provides methods that encapsulate the following queries:

Lower Level Components Owned by a Given Object Using the ownership hierarchy, NeuroArch can easily retrieve the tree of lower level components owned by a specified object (or some portion thereof) up to some arbitrary number of ownership levels. These components may in turn be used to obtain the induced subgraph if there exist data transmission edges between those components. This functionality

 DOI:10.5281/zenodo.44225

facilitates extraction of subcircuits from the biological or executable circuit data stored in NeuroArch. For example, the subgraph of `Neuron` and `Synapse` instances for a specified `Neuropil` instance may be obtained given the latter:

```
# Extract node corresponding to lamina neuropil; the 'graph' object
# encapsulates the entire graph database:
lamina = graph.neuropils.query(name='lamina').one()

# Find subgraph of neurons and synapses:
result = lamina.traverse_owns(['Neuron', 'Synapse'])
```

Higher Level Components that Own a Given Object By traversing the ownership hierarchy from lower level components to higher level components, NeuroArch can determine what high-level biological subdivisions or executable circuit abstractions contain a given component. For example, one may determine which `Cartridge` instance in the lamina neuropil owns a given L1 neuron represented by an `Neuron` instance.

Multicriterion Filtering of Query Results Low-level graph query languages can be used to easily extract classes of elements or elements with specific attribute values; restricting those queries to the results of traversals that return subgraphs corresponding to biological or executable circuit motifs increases the complexity of the queries required to obtain the desired results. To address this increase in complexity, NeuroArch enables the results of a query to be qualified by simultaneous application of multiple search criteria. For example, all neuron membrane models of L2 neurons in the lamina with a specific model parameter value can be extracted as follows:

```
# Extract node corresponding to lamina LPU; the 'graph' object
# encapsulates the entire graph database:
lamina = graph.LPUs.query(name='lamina').one()

# Find subgraph of neuron membrane model instances
# and synapse model instances:
lamina_ml = lamina.traverse_owns(['MembraneModel', 'SynapseModel'])

# Restrict query to Morris-Lecar instances
# modeling L1 neurons with a specific parameter value:
result = lamina_ml.has(attrs={'name': 'L1', 'phi': 0.025},
                       classes=['MorrisLecar'])
```

5.3 Support for Operations on Query Results

NeuroArch supports the passing of OGM query results to graph operators to enable intuitive expression of complex queries in terms of set operations such as union, intersection, and difference applied to the nodes in a subgraph. As an example, the difference operator can be used to exclude all amacrine cells from the lamina LPU circuit. If the original lamina circuit comprises executable components supported by Neurokernel, the modified circuit may also be executed.

```
# Extract node corresponding to lamina LPU:
lamina = graph.lpus.query(name='lamina').one()

# Extract all nodes corresponding to specific neuron membrane potential
# or conductance-based synapse models:
all_lamina_neuron_synapses = \
    lamina.traverse_owns(['MorrisLecar','ConductanceSynapseModel'])

# Find all amacrine neurons by name:
amacrine_neurons = all_lamina_neurons.has(attrs={'name': 'Am'})

# Obtain subgraph determined by difference of nodes:
lamina_without_amacrine = all_lamina_neurons_synapses \
                        - amacrine_neurons
```

5.4 Multimodal Views

NeuroArch's OGM provides access to object or query result data in views that expose both tabular and graph data structures to support different applications. NeuroArch uses the tabular and graph data structures respectively provided by Pandas[5] [17] and NetworkX[6] [13]; this enables use of the rich APIs provided by these actively developed and widely used packages to access and/or manipulate exposed data. Multimodal views are both readable and writable; NeuroArch can propagate modifications made to data exposed by a view back into its database. Since NeuroArch exposes the results of a query performed through its OGM, a view to the results of a query can therefore seamlessly expose the data associated with multiple nodes or edges returned by the query within a single tabular or graph data structure.

To illustrate the utility of multimodal views, consider the scenario of modifying a particular parameter in all model instances of a particular neuron type in a model of the lamina LPU. By exposing the model parameters of all instances of the neuron

[5] http://pandas.pydata.org
[6] http://networkx.github.io

type in question as a Pandas `DataFrame` object, the object's API may be exploited to perform the desired modification with a single line of code:

```
# Extract node corresponding to lamina LPU:
lamina = graph.lpus.query(name='lamina').one()

# Extract all nodes corresponding to specific neuron
# membrane potential model:
all_lamina_neurons = \
    lamina.traverse_owns(['MorrisLecar'])

# Find all L1 neurons by name:
L1_neurons = all_lamina_neurons.has(attrs={'name': 'L1'})

# Set phi parameter of all L1 neurons to single value:
L1_neurons.view_table['phi'] = 0.03

# Save modifications to view:
L1_neurons.view_table_save()
```

One can visualize the graph structure of the query results by exposing the same query as a NetworkX graph:

```
import networkx as nx

# Convert graph to pygraphviz format and set visualization attributes:
g = L1_neurons.view_graph()
p = nx.to_agraph(g)
p.node_attr.update({'shape': 'rect', 'style': 'filled'})
p.draw('L1_neurons.jpg', prog='circo')
```

5.5 Interface to Neurokernel

To enable evaluation of stored circuit models, NeuroArch's API can be invoked directly by a Neurokernel emulation to instantiate and execute circuits stored in NeuroArch's database. Circuit models stored in NeuroArch can only be executed if they comprise components with numerical implementations provided by Neurokernel. Since the graph structure of LPU circuit data used by the implementation of Neurokernel described in [9] differs from that assumed by NeuroArch's data model (§ 3.3), NeuroArch provides graph transformation routines for converting extracted data to the structure expected by Neurokernel. The latter routines will become unnecessary when Neurokernel is updated to be directly compatible with NeuroArch's data model.

6 Testing Neuroarch's Functionality

To test the features described in § 5, NeuroArch was used to address the following proof-of-concept scenarios.

Arbitrary LPUs consisting of about 100 non-spiking Morris-Lecar neurons and Leaky Integrate-and-Fire neurons randomly connected with alpha-function and conductance-based synapses were generated using NetworkX and loaded into NeuroArch's database along with connectivity patterns that linked random ports exposed by each LPU. The LPU and pattern generation algorithm was identical to that provided in the introductory example included in the Neurokernel repository [7]. NeuroArch's OGM was invoked within a Neurokernel emulation to extract these LPUs and pattern circuits, convert them to the current graph structure expected by Neurokernel (§ 5.5), and instantiate the object classes required to execute the emulation. The output was successfully validated for a simple input signal provided to the same neurons in both the introductory example and the NeuroArch example.

To examine more realistic circuit scenarios, we scaled up the above scenario by increasing (i) the number of LPUs (up to 8 LPUs), (ii) the number of neurons within each LPU (up to 10,000 per LPU), and (iii) the number of ports exposed by each LPU (up to 10,000 per LPU). We also loaded, extracted, and executed a lamina/medulla model comprising almost 17,000 neurons developed for Neurokernel testing purposes [8]. NeuroArch was able to handle all of these scenarios, although the time required to both load LPU data into NeuroArch's database and retrieve it within a Neurokernel emulation increased noticeably with the total number of components in the overall circuit due to the nonoptimal configuration of the database and system used by NeuroArch.

Finally, we used NeuroArch's multimodal views to modify the parameters of select populations of neurons and synapses within the above LPU circuits prior to extraction and execution by Neurokernel. We validated the effects of these modifications by recording the expected perturbations of the activity of the spiking neurons of the example LPUs and the graded potential neurons in the lamina/medulla model.

7 Summary and Future Plans

NeuroArch aims to accelerate the cycle of neural circuit modeling, design, evaluation, and biological validation [9]. The requirements delineated in this RFC describe a

[7] http://github.com/neurokernel/neurokernel/tree/master/examples/intro
[8] http://github.com/neurokernel/vision

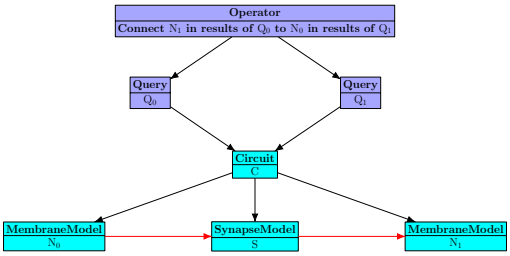
 DOI:10.5281/zenodo.44225

minimal architecture for achieving this aim in conjunction with the model execution support of Neurokernel. This section describes several future areas for improvement and extension of NeuroArch's architecture.

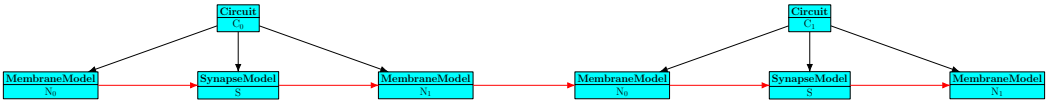
7.1 Model Construction Using Composition Operations

A major advantage of NeuroArch's OGM (§ 5.1) is that it enables the result of a query on existing neural circuit data to be treated as an operand that can be manipulated by query result operators. Given that existing anatomical datasets (such as that of the medulla from Janelia [2]) provide incomplete data regarding the structure of neuropils, the process of inferring circuit functionality could exploit NeuroArch's encapsulation of query results and support for operators defined on those results to construct more comprehensive circuit models by composing subunits that each consist of the result of individual queries. Although such circuits can be stored in NeuroArch's database by fully expanding the operators and their operands into a graph of components comprised by the current NeuroArch data model, doing so does not store any information as to how the circuit is defined in terms of subgraphs and operators.

To store the latter information, NeuroArch's data model could be extended to introduce nodes that correspond to operators and query results, the latter which own the component nodes extracted by the query. This would enable storage of the execution tree of operator and query result nodes that must be processed to obtain the constructed circuit (Fig. 7). To obtain the fully equivalent graph of low-level objects corresponding to the representation in terms of query results and operators, NeuroArch's query API would need to provide services that can execute the operators stored in the database.



(a) Representation of sample circuit in terms of operators and queries (blue) and the components (cyan) comprised by an individual query. In this example, both queries return the same subgraph of components.



(b) Equivalent components of sample circuit described by queries and operators in Fig. 7a.

Figure 7: Example of how a circuit may be defined in terms of graph operators applied to query results and the motifs extracted by individual queries. As in Figs. 2 and 3, black edges denote ownership while red edges denote data transmission connections between objects.

7.2 Using NeuroArch Data for Neurokernel Resource Allocation

From a performance perspective, executable circuits stored in NeuroArch could be analyzed to estimate the computational resources required by Neurokernel to efficiently run a given circuit architecture on available GPU resources. For example, the graph of a circuit's constituent neurons and synapses could be processed by a graph partitioning algorithm to determine how to amortize data transmission costs between GPUs during execution. To enable the above functionality, NeuroArch's API would need to provide services for extracting relevant circuit information required to compute resource requirements. NeuroArch's data model could also be extended to explicitly include metadata regarding the computational costs of different executable elements in its database. For example, an instance of a point model of a neuron's membrane potential might be assigned a higher cost than an instance of a passive multicompartmental model.

7.3 Support for Input/Output File Formats

NeuroArch should support loading data from and saving data to several specification formats such as SWC[9], CSV, GEXF, NeuroML [10], NineML [20], or SpineML [21] to

(i) facilitate importing of data in those formats into NeuroArch for use in circuit design,

(ii) enhance interoperability with other tools that employ those formats, and

(iii) enable sharing of data between users running different NeuroArch instances (§ 7.4).

Some of this functionality can be achieved by exploiting the import/export features of the Python packages used by NeuroArch's multimodal views that support some of the above formats (§ 5.4). Loading/saving of multiple versions of a single model should also be supported. Currently available neural circuit datasets that are in a non-standard format (such as the medulla data from Janelia, manually constructed annotations for a specific dataset, etc.) will require customized loaders; NeuroArch's API should expose Python functions and/or classes that must be used by a new data loader to manipulate the database. This part of the API should manage creation of new nodes and relationships in the database, handle versioning, and perform requisite sanity checks to prevent inadvertent loading of incorrectly formatted data.

Given that NeuroArch affords researchers with the opportunity to define entirely new modeling elements and architectural abstractions (§ 3.3) support for import/-export of a model data specified using components defined in a fixed schema (such as that of NeuroML) necessarily limits what sort of abstractions may be represented in an imported/exported model specification. This limitation could be addressed by generation/parsing of customized XML schemas alongside exported/imported models; NeuroML's parser generation mechanism (which is used by Neurokernel's current support for importing NeuroML-like XML) can be exploited to address this need.

In the event that NeuroArch is extended to support storage of model execution state snapshots (§ 7.8), its data sharing services should also be extended to provide a way to store/load such data in a suitable file format.

[9] http://www.neuronland.org/NLMorphologyConverter/MorphologyFormats/SWC/Spec.html

7.4 Online Data Sharing

To facilitate sharing of models with other researchers, NeuroArch should provide a service whereby biological or circuit design data stored in one NeuroArch instance can be easily shared with other researchers. This could be achieved either

(i) by enabling loading/saving of an entire model in a suitable file format; (§ 7.3);

(ii) by enabling running NeuroArch instances to expose services on the Internet that permit them to be queried (which should be technically possible given that the underlying OrientDB graph database supports network access); or

(iii) by providing a service that enables models to be easily published online in a form that can be immediately imported into other NeuroArch instances. This service could potentially

  (a) use a revision control system such as Git or Mercurial to upload data to or retrieve data from a public repository on GitHub[10] or Bitbucket[11];

  (b) take advantage of the API provided by the Zenodo research data sharing service[12] to automatically request a DOI for a published model that could be made available to other researchers as the access point for obtaining model data for immediate loading into a NeuroArch instance.

7.5 Performance Assessment

Given that complex queries performed by NeuroArch's OGM can be computationally intensive, there is a need to quantify the performance of NeuroArch's query mechanism and graph operator support in a range of circumstances to optimize future performance. NeuroArch should therefore provide a means of benchmarking the data access and manipulation services provided by its API.

7.6 Graphical/Visualization Frontend

Combining biological and circuit design data from multiple sources in a graph database opens the doors to the development of new user applications for interacting with fly brain data. While such applications should be developed independently of the core NeuroArch software, the NeuroArch API must ensure that queries required by such frontend applications will be supported.

[10] http://github.com
[11] http://bitbucket.org
[12] http://zenodo.org/dev

7.7 Support for Dynamic Models

Model configurations executed by Neurokernel cannot currently change during execution, i.e., the projected flow of model data from NeuroArch to Neurokernel is unidirectional. This effectively precludes development of models whose parameters or structure change over the course of model execution. Apart from enabling consideration of a new class of circuit models, support for dynamically changing stored model data could be useful in developing semiautomated model refinement systems. To support model plasticity, NeuroArch's API must provide low-latency services for propagating updates to a stored model's parameters in real-time without degrading the performance of model execution by Neurokernel.

7.8 Storing Model States

Software debuggers provide programmers with the means of examining variable states at times prior to termination of program execution to pinpoint the causes of anomalous program behavior. The analogous ability to obtain a snapshot of a circuit model's states at points during execution by Neurokernel before a model has finished running is similarly valuable to model refinement. NeuroArch's data model could be extended to support representation of state data associated with the components of an executed circuit model at multiple times.

8 Acknowledgements

This work was supported in part by the AFOSR under grant #FA9550-12-10232 and in part by the NSF under grant #1544383. The authors wish to thank Prof. Brian McCabe (Dept. of Neuroscience, Columbia University) for useful discussions and feedback.

References

[1] Brainbase. http://brainbase.imp.ac.at/. Accessed: 2015-12-01.

[2] Fruit fly medulla connectome. https://github.com/janelia-flyem/ConnectomeHackathon2015. Janelia Research Campus, HHMI. Accessed: 2015-12-01.

[3] The Open Connectome Project. http://www.openconnectomeproject.org. Accessed: 2015-12-01.

[4] The Neuroscience Information Framework: A Data and Knowledge Environment for Neuroscience. Neuroinformatics, 6(3):149–160, October 2008. http://link.springer.com/article/10.1007/s12021-008-9024-z.

[5] J. D. Armstrong, K. Kaiser, A. Müller, K. F. Fischbach, N. Merchant, and N. J. Strausfeld. Flybrain, an on-line atlas and database of the Drosophila nervous system. Neuron, 15(1):17–20, July 1995. http://flybrain.neurobio.arizona.edu/.

[6] Giorgio A. Ascoli, Duncan E. Donohue, and Maryam Halavi. NeuroMorpho.Org: A Central Resource for Neuronal Morphologies. 27(35):9247–9251, August 2007. http://www.jneurosci.org/content/27/35/9247.short.

[7] Marta Costa, Simon Reeve, Gary Grumbling, and David Osumi-Sutherland. The Drosophila anatomy ontology. Journal of Biomedical Semantics, 4(1):32, October 2013. http://www.jbiomedsem.com/content/4/1/32/abstract.

[8] Gilberto dos Santos, Andrew J. Schroeder, Joshua L. Goodman, Victor B. Strelets, Madeline A. Crosby, Jim Thurmond, David B. Emmert, William M. Gelbart, and the FlyBase Consortium. FlyBase: introduction of the Drosophila melanogaster Release 6 reference genome assembly and large-scale migration of genome annotations. Nucleic Acids Research, November 2014. http://nar.oxfordjournals.org/content/early/2014/11/14/nar.gku1099.

[9] Lev E. Givon and Aurel A. Lazar. Neurokernel: An open source platform for emulating the fruit fly brain. PLOS ONE, January 2016. In press. Also available as Neurokernel RFC #4: http://dx.doi.org/10.5281/zenodo.31947.

[10] Padraig Gleeson, Sharon Crook, Robert C. Cannon, Michael L. Hines, Guy O. Billings, Matteo Farinella, Thomas M. Morse, Andrew P. Davison, Subhasis Ray, Upinder S. Bhalla, Simon R. Barnes, Yoana D. Dimitrova, and R. Angus Silver. NeuroML: a language for describing data driven models of neurons and networks with a high degree of biological detail. PLOS Computional Biology, 6(6):e1000815, June 2010. http://dx.doi.org/10.1371/journal.pcbi.1000815.

[11] Padraig Gleeson, Eugenio Piasini, Sharon Crook, Robert Cannon, Volker Steuber, Dieter Jaeger, Sergio Solinas, Egidio D'Angelo, and R. Angus Silver. The Open Source Brain Initiative: enabling collaborative modelling in computational neuroscience. BMC Neuroscience, 13(Suppl 1):O7, July 2012. http://www.biomedcentral.com/1471-2202/13/S1/O7/.

[12] Padraig Gleeson, Volker Steuber, and R. Angus Silver. neuroConstruct: a tool for modeling networks of neurons in 3D space. Neuron, 54(2):219–235, April 2007. http://dx.doi.org/10.1016/j.neuron.2007.03.025.

[13] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In Gäel Varoquaux, Travis Vaught, and Jarrod Millman, editors, Proceedings of the 7th Python in Science Conference (SciPy2008), pages 11–15, August 2008. http://conference.scipy.org/proceedings/SciPy2008/index.html.

[14] Michael L. Hines, Thomas Morse, Michele Migliore, Nicholas T. Carnevale, and Gordon M. Shepherd. ModelDB: A Database to Support Computational Neuroscience. Journal of Computational Neuroscience, 17(1):7–11, August 2004. 10.1023/B:JCNS.0000023869.22017.2e.

[15] Kei Ito, Kazunori Shinomiya, Masayoshi Ito, J. Douglas Armstrong, George Boyan, Volker Hartenstein, Steffen Harzsch, Martin Heisenberg, Uwe Homberg, Arnim Jenett, Haig Keshishian, Linda L. Restifo, Wolfgang Rössler, Julie H. Simpson, Nicholas J. Strausfeld, Roland Strauss, Leslie B. Vosshall, and Insect Brain Name Working Group. A systematic nomenclature for the insect brain. Neuron, 81(4):755–765, February 2014. http://www.cell.com/neuron/abstract/S0896-6273(13)01178-1.

[16] Aurel A. Lazar, Nikul H. Ukani, and Yiyin Zhou. The cartridge: A canonical neural circuit abstraction of the lamina neuropil – construction and composition rules. Neurokernel Request for Comments, Neurokernel RFC #2, January 2014. http://dx.doi.org/10.5281/zenodo.11856.

[17] Wes McKinney. pandas: a foundational python library for data analysis and statistics. In International Conference for High Performance Computing, Networking, Storage, and Analysis, November 2011. http://www.scribd.com/doc/71048089/pandas-a-Foundational-Python-Library-for-Data-Analysis-and-Statistics.

[18] Nestor Milyaev, David Osumi-Sutherland, Simon Reeve, Nicholas Burton, Richard A Baldock, and J. Douglas Armstrong. The Virtual Fly Brain browser and query interface. Bioinformatics, 28(3):411–415, February 2012. http://bioinformatics.oxfordjournals.org/content/28/3/411.

[19] Pablo Pareja-Tobes, Raquel Tobes, Marina Manrique, Eduardo Pareja, and Eduardo Pareja-Tobes. Bio4j: a high-performance cloud-enabled graph-based data

platform. bioRxiv, page 016758, March 2015. http://dx.doi.org/10.1101/016758.

[20] Ivan Raikov and INCF Multiscale Modeling Taskforce. NineML - a description language for spiking neuron network modeling: the abstraction layer. In BMC Neuroscience 2010, volume 11, page P66, San Antonio, 2010. http://www.biomedcentral.com/1471-2202/11/S1/P66.

[21] Paul Richmond, Alex Cope, Kevin Gurney, and David J. Allerton. From Model Specification to Simulation of Biologically Constrained Networks of Spiking Neurons. Neuroinformatics, 12(2):307–323, November 2013. http://dx.doi.org/10.1007/s12021-013-9208-z.

[22] Marko A. Rodriguez. The Gremlin Graph Traversal Machine and Language (Invited Talk). In Proceedings of the 15th Symposium on Database Programming Languages, DBPL 2015, pages 1–10, New York, NY, USA, 2015. ACM. http://doi.acm.org/10.1145/2815072.2815073.

[23] Johannes Sorger, Katja Buhler, Florian Schulze, Tianxiao Liu, and Barry Dickson. neuroMAP - interactive graph-visualization of the fruit fly's neural circuit. In 2013 IEEE Symposium on Biological Data Visualization (BioVis), pages 73–80, 2013. http://dx.doi.org/10.1109/BioVis.2013.6664349.